

**Angerhausen · Brückmann**  
**Englisch · Gerits**

# **64** **intern**

**Das große Buch zum**  
**COMMODORE 64**  
**mit dokumentiertem Schaltplan**

**EIN DATA BECKER BUCH**



**Angerhausen · Brückmann**  
**Englisch · Gerits**

# **64** **intern**

**Das große Buch zum**  
**COMMODORE 64**  
**mit dokumentiertem Schaltplan**

**EIN DATA BECKER BUCH**

ISBN 3-89011-000-2

3. erweiterte und überarbeitete Auflage

Copyright (C) 1983 DATA BECKER GmbH  
Merowingerstr. 30  
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.



### Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden. Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.



## V o r w o r t

Der COMMODORE 64 ist eine SUPERMASCHINE. Das war uns schon nach kurzer Arbeit mit dem Gerät klar. Selbst mit dem besten Computer kann man jedoch herzlich wenig anfangen, wenn man nichts über seine Funktionen und seine Bedienung weiß.

Hier war der einzige Punkt, in dem uns der neue COMMODORE 64 enttäuschte. Das sehr mageres Handbuch war in Englisch, und weiterführende Literatur war nicht einmal in den USA zu bekommen.

Also entschlossen wir uns, selbst ein Buch zu schreiben. Das Ergebnis liegt vor Ihnen. Es erhebt keinen Anspruch auf Vollständigkeit oder auf schriftstellerische Qualitäten. Dafür enthält dieses Buch die umfassenden Ergebnisse monatelanger Kleinarbeit. Wir hoffen, daß es Ihnen hilft, die hervorragenden Qualitäten des Commodore 64 auszunutzen.

Die Autoren

Michael Angewissen

U. Kuhn

W. Engelke

V. C. - D. J.

## V o r w o r t zur zweiten Auflage

Nach nur 2 1/2 Monaten war die erste Auflage von 64 INTERN bereits vergriffen. Die jetzt vorliegende zweite Auflage wurde komplett überarbeitet und in einigen Kapiteln erweitert.

Für die vielen Anregungen und Tips aufmerksamer Leser dürfen wir ganz herzlich danken.

Düsseldorf, im April 1983

Die Autoren

### Vorwort zur 3. erweiterten und überarbeiteten Auflage

Auch die zweite Auflage von 64 INTERN war innerhalb kurzer Zeit vergriffen. Über 15.000 verkaufte Exemplare und die große Resonanz aus unserem Leserkreis bestätigte nicht nur die Richtigkeit der Konzeption dieses Buches, sondern gab uns auch wieder viele Anregungen und Tips, für die wir ganz herzlich danken. Wir haben deshalb die Gelegenheit der Neuauflage benutzt, um das Buch in wesentlichen Teilen zu erweitern und zu überarbeiten.

Zu allen Maschinenprogrammen und -routinen dieses Buches finden Sie jetzt zusätzlich einen BASIC-Loader. Mit Hilfe dieser einfachen BASIC-Programme können jetzt auch alle Nicht-Maschinenprogrammierer die Programme in Maschinensprache leicht eingeben und voll nutzen. Der jeweilige BASIC-Loader überprüft das Programm auch auf Eingabefehler.

Besonders danken möchten wir der Firma COMMODORE in Frankfurt, die uns den Abdruck der Original-Schaltpläne ermöglichte. Wir haben es aber nicht beim Abdruck der Schaltpläne belassen, sondern zusätzlich eine 30-seitige Dokumentation hinzugefügt, die die Schaltpläne exakt beschreibt, erklärt und auch dem technisch interessierten Laien zugänglicher macht.

An der dritten Auflage dieses Buches waren beteiligt:

Michael Angerhausen, 20 Jahre,  
DATA BECKER Softwarespezialist

Rolf Brückmann, 27 Jahre,  
DATA BECKER Hardwarespezialist

Lothar Englisch, 28 Jahre,  
DATA BECKER Softwarespezialist

Klaus Gerits, 34 Jahre,  
Leiter der DATA BECKER Entwicklungsabteilung

Düsseldorf, im August 1983

## INHALTSVERZEICHNIS

### Kapitel 1 : DER COMMODORE 64 UNTER DIE LUPE GENOMMEN

1.1	Das sollten Sie über das Gerät wissen .....	7
1.2	Realisierung der Hardware .....	9
1.3	Besonderheiten des 6510 Prozessors .....	10
1.4	Speicherbelegungspläne .....	11
1.5	Die Erweiterungsschnittstelle .....	22
1.6	Der User-Port .....	23
1.6.1	Handhabung des User-Ports in BASIC .....	24

### Kapitel 2 : DER SYNTHESIZER UND SEINE PROGRAMMIERUNG

2.1	Der Sound-Controller 6581 .....	27
2.1.1	Allgemeines über den 6581 - Blockschema .....	27
2.1.2	Registerbeschreibung des SID .....	30
2.1.3	Der Analog / Digital-Wandler .....	32
2.1.3.1	Die Handhabung des A/D-Wandlers .....	33
2.1.3.2	Die Verwendung von Paddles .....	33
2.2	Die Programmierung des 6581 .....	35
2.3	SYNTHY-64 .....	38

### Kapitel 3 : DIE GRAPHIK UND IHRE PROGRAMMIERUNG

3.1	Der Video-Controller 6569 VIC .....	41
3.1.1	Allgemeines über den 6569 .....	41
3.1.2	Registerbeschreibung des VIC .....	42
3.1.3	Die Betriebsarten des VIC .....	43
3.2	Die Schnittstelle zum Prozessor .....	45
3.3	Die Schnittstelle zum RAM - Blockschema .....	46
3.4	Zeichengenerator-Schnittstelle - Blockschema .....	46
3.5	Color-RAM-Schnittstelle - Blockschema .....	48
3.6	Programmierung von Farbe und Graphik .....	50
3.7	SPRITES - Das Zauberwort auf dem Commodore 64 .....	64
3.7.1	Programmierung der SPRITES .....	66

### Kapitel 4 : EIN- AUSGABESTEUERUNG - CIA 6526

4.1	Allgemeines über den 6526 .....	76
4.2	Register-Plan .....	76
4.3	E/A-Ports .....	80
4.4	Timer .....	81
4.5	Die Echtzeituhr .....	82
4.5.1	Mit einem Trick die richtige Zeit .....	82
4.6	Die CIAs im Commodore 64 .....	83
4.7	Die Verwendung von Joy-Sticks .....	85

### Kapitel 5 : BASIC EINMAL ANDERS GESEHEN

5.1	So arbeitet der BASIC Interpreter .....	86
5.2	Von der Eingabe bis zur Verarbeitung .....	87
5.3	Machen Sie mehr aus Ihrem BASIC .....	88
5.3.1	Wie erweitere ich BASIC .....	88
5.3.2	HARDCOPY - RE-NEW - PRINT USING .....	88
5.3.3	Mathematische Routinen selbst entwickelt .....	93
5.3.4	SQR-Funktion - SUM Funktion .....	94
5.3.5	Umwandlung Fließkomma - Integer - Byte Format .....	97

## Kapitel 6 : MASCHINENPROGRAMMIERUNG AUF DEM COMMODORE 64

6.1	Der Monitor - Und was steckt dahinter .....	100
6.2	Nützliche Adressen des Commodore 64 ROM .....	104
6.3	Ein- und Ausgaberroutinen im Betriebssystem .....	107
6.3.1	Festlegung des Ein- Ausgabegerätes .....	107
6.3.2	Ein- und Ausgabe von einzelnen Bytes .....	108
6.3.3	LOAD und SAVE - Die Technik der Datenspeicherung ..	110
6.3.4	Die Programmierung der RS232 .....	115
6.3.5	Der serielle IEC-BUS .....	118

## Kapitel 7 : VC-20 - COMMODORE 64 - CBM

7.1	Die Belegung der Zeropage .....	121
7.2	Die Adressen der BASIC-Routinen .....	125
7.3	Vergleichstabelle VC-20 - Commodore 64 .....	129
7.4	Vergleichstabelle Commodore 64 - CBM 8000 .....	131
7.5	Umsetzung von VC-20 Programmen auf Commodore 64 ...	134
7.6	Umsetzung von CBM Programmen auf Commodore 64 .....	135

## Kapitel 8 : COMMODORE 64 - DAS ROM LISTING

8.1	Die Nutzung des ROM-Listings .....	136
8.2	Verzeichnis der ROM-Routinen .....	138
8.3	Das große ROM-Listing .....	144

Kapitel 9 : DOKUMENTATION ZUM COMMODORE 64 SCHALTPLAN	284
---	-----

## ANHANG

## E I N F Ü H R U N G

Der Commodore 64 setzt neue Maßstäbe im Preis-/Leistungsverhältnis. Zum Preis eines Heimcomputers bietet er durchaus professionelle Qualität. Als Commodore 64 Besitzer hat man praktisch mehrere Computer in einem.

### DER LERN- UND EINSTIEGSCOMPUTER

Sein niedriger Preis qualifiziert ihn durchaus als Einstiegsgerät. Zusammen mit der Datassette bildet der Commodore 64 bereits in der Grundversion ein sehr leistungsfähiges Computersystem. Wichtig ist vor allem, daß der Einsteiger nicht mit wachsenden Kenntnissen stets an Grenzen des Gerätes stößt.

Leicht gemacht wird das Lernen auch durch das umfangreiche und komfortable Commodore BASIC. Die sehr weitgehende Kompatibilität des Commodore 64 BASIC zu den BASIC-Interpretern der anderen Commodore-Computer erleichtert die Übernahme von Programmen und erschließt somit eine sehr große Programmbibliothek.

'64 intern' ist zwar nicht für den unmittelbaren Anfänger geschrieben, doch bildet das beim Commodore 64 ohnehin dürftige Handbuch in der Regel höchstens in den ersten Wochen eine ausreichende Lektüre. Wer tiefer einsteigen möchte als dies das Handbuch ermöglicht, sollte in unserem Buch zunächst mit den Kapiteln 2 (Synthesizer) und 3 (Graphik) beginnen. Wichtig ist auch das Kapitel 7, da die dort angegebenen Hinweise helfen, für andere Rechner geschriebene Programme mit dem Commodore 64 zu nutzen.

### DER SPEICHERRIESE

Besonders fortgeschrittene Programmierer werden den großen Speicherraum des Commodore 64 zu schätzen wissen. Mit 64 K RAM und 20 K ROM bietet der Commodore 64 genügend Speicherreserven auch für das anspruchsvollste Programm. Die Speicherbelegung des Commodore 64 in den verschiedenen

möglichen Betriebsformen haben wir deshalb im Kapitel 1 ausführlich dargestellt.

Besonders gut lassen sich die Speicherreserven des Commodore 64 und viele seiner fortschrittlichen Möglichkeiten in Maschinensprache nutzen. Um auch reine BASIC-Programmierer zu einem Einstieg in die Maschinenprogrammierung zu ermutigen, haben wir unserem Buch eine Einführung in Maschinen- und Assemblerprogrammierung vorangestellt.

Maschinenprogrammierer können sich selbst noch das Arbeiten in BASIC wesentlich erleichtern und noch komfortabler machen. Wir haben im Kapitel 5 beschrieben, wie man neue BASIC-Befehle einbindet. Sicherlich wird es Ihnen nach einigem Experimentieren nicht schwerfallen, ähnlich den beschriebenen Beispielbefehlen eigene, neue Befehle zu erstellen. Wichtige Hilfestellung dabei und bei allen anderen Maschinenprogrammen kann nicht nur das ausführlich dokumentierte Listing des Commodore 64 Betriebssystems leisten, sondern auch die Beschreibung der Funktion des BASIC Interpreters.

#### DER GRAPHIKCOMPUTER

Wo bei vielen anderen Computern erst mühsam durch teure Hardware-Zusätze grafische Fähigkeiten nachgerüstet werden müssen, bietet der Commodore 64 von Hause aus bereits serienmäßig besondere Fähigkeiten. Den Schlüssel zur hochauflösenden Farbgraphik bildet ein völlig neuer Videocontroller, dessen Eigenschaften und Fähigkeiten wir eingehend beschrieben haben.

Detailliert sind wir auch auf ein neues Zauberwort eingegangen: SPRITES. Diese supergroßen, frei definierbaren Graphikzeichen machen vom Action-Spiel bis zur werbewirksamen Laufschrift Dinge möglich, die bisher, wenn überhaupt, nur unter großem Programmieraufwand realisiert werden konnten.

Die überragenden Farbgraphikmöglichkeiten des Commodore 64 und der niedrige Preis lassen erwarten, daß der Commodore 64 demnächst auch als intelligenter Farbprozessor für andere,



größere Anlagen Verwendung findet. Nötig ist hierzu lediglich eine hardwareseitige Verbindung, z.B. über den seriellen Bus oder einen IEC-Bus. Auch für andere Zwecke ließe sich der Commodore 64 nach ähnlichem Muster gut als Hilfsrechner einsetzen, z.B. als Datenbankprozessor.

### DER SYNTHESIZER

Im Commodore 64 ist Musik!

Als einer der ersten Kleincomputer verwendet der Commodore 64 einen vollwertigen Synthesizer, der sich hinter professionellen Spezialgeräten nicht zu verstecken braucht. Das Geheimnis ist hier auch wieder ein völlig neu entwickelter Chip, dessen Fähigkeiten und Möglichkeiten wir für Sie genauestens unter die Lupe genommen haben.

Eigentlich fehlt nur noch eine entsprechende Klaviertastatur, um aus dem Commodore 64 eine Heimorgel zu machen, die mit entsprechender Programmierung viele dieser erheblich größeren und teureren Geräte in den Schatten stellt. Es würde uns nicht wundern, wenn COMMODORE zum Commodore 64 demnächst eine solche Tastatur anbietet.

Eine andere, interessante Möglichkeit wäre der Anschluß des Commodore 64 an eine Stereoanlage. Musikliebhaber werden am Commodore 64 ihre helle Freude haben und ihre Ohren werden Augen machen.

### DER STEUERUNGSCOMPUTER

Auch für Steuerungs- und Regelungszwecke der unterschiedlichsten Art läßt sich der Commodore 64 mit entsprechenden Interfaces gut verwenden. Sowohl der engagierte Hobby-Elektroniker als auch der industrielle Anwender, für den der Commodore 64 ungewöhnlich viel Computerleistung zu diesem Preis bietet, werden rasch entsprechende Anwendungsgebiete finden. Ein in diesem Buch beschriebenes Beispiel für solche Anwendungen ist die Benutzung des Paddleports als A/D-Wandler. Weitere Hinweise gibt die Beschreibung der Eingabe-/Ausgabe-Steuerung im

Kapitel 4 und der entsprechenden Betriebssystemroutinen im 6. Kapitel.

#### IHR EIGENER COMPUTER

Ganz gleich, ob Sie den Commodore 64 gekauft haben, um damit zu komponieren, Adressen zu speichern oder um programmieren zu lernen, '64 intern' wird Ihnen helfen, die fantastischen Möglichkeiten des Commodore 64 zu erschließen.

Bestimmt finden Sie Anwendungsgebiete für Ihren eigenen Computer, an die Sie bis heute nicht gedacht haben.

Wer noch tiefer einsteigen möchte, findet im letzten Kapitel eine Auflistung empfehlenswerter Mikrocomputer-Literatur.

## Kapitel 1 : DER CBM 64 UNTER DIE LUPE GENOMMEN

### 1.1 Das sollten Sie über das Gerät wissen

Der CBM 64 kann ohne weiteres als Bilanzleistung angesehen werden, was das Bemühen betrifft, mit relativ geringem Aufwand ein hervorragendes Produkt zu schaffen.

Sie werden bei Ihrer Arbeit mit dem Gerät sehr schnell feststellen, daß eigentlich, was die Hardware betrifft, an keiner Ecke etwas fehlt.

Wenn Sie auch Besitzer eines VC-20 sind, sollten Sie sich einmal die Mühe machen, beide Geräte zu öffnen.

Sie werden feststellen, daß der CBM 64 weniger ICs enthält, trotz seiner höheren Leistungsfähigkeit.

Dies ist nur durch eine höhere Integrationsdichte der ICs möglich.

In der Tat stecken im CBM 64 eine Reihe neuentwickelter, hochintegrierter ICs.

Die Firma Commodore ist in nämlich in der glücklichen Lage, über eine eigene Halbleiterfabrikation in Form der Tochterfirma MOS zu verfügen.

Schauen Sie sich das Foto vom Innenleben des CBM 64 an. Sie werden es nicht für möglich halten, daß dies einen kompletten Single Board Computer (Einplatinenrechner) einschließlich sämtlicher zur Kommunikation mit der Außenwelt erforderlicher Elemente darstellt.

Es ist dem Hersteller gelungen, auf 64-K-Adressraum, die ein 8-Bit-Mikroprozessor für gewöhnlich aufweist, folgende Dinge unterzubringen:

- \* 64K dynamischer RAM
- \* 1K Farb-RAM
- \* 4K Charactergenerator
- \* Farbvideocontroller mit Hi-Res-Graphik
- \* Synthesizer mit drei unabhängigen Stimmen
- \* 8K Basic
- \* 8K Betriebssystem
- \* 2 Parallel-I/O

Das Blockschema des CBM 64 finden Sie auf der nächsten Seite.

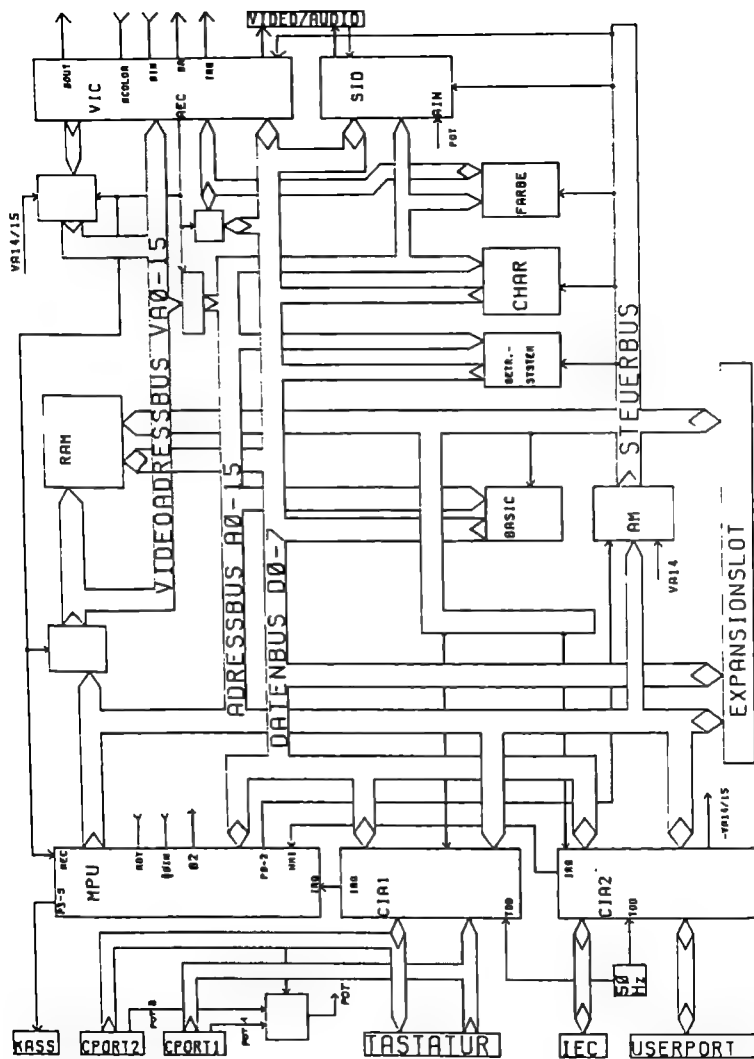
Auf einige Einzelheiten daraus wird noch im Kapitel 3 und im Abschnitt 1.4 näher eingegangen.

Sicher werden Sie sich fragen, wie das alles untergebracht werden kann, so daß es auch noch funktioniert, wo doch alleine der Ram schon den gesamten zur Verfügung stehenden Adressraum einnimmt.

Da die in den folgenden Abschnitten erklärten Details notwendigerweise sehr technischer Natur sein müssen, ist die Lektüre allgemein in die Mikrocomputertechnik einführender Literatur empfehlenswert.

Der hier behandelte Themenkreis kann insbesondere durch die Bücher von MOS vertieft werden.

Es sind dies das Hardware-Handbuch und das Programmierhandbuch, die sich beide speziell auf die ICs der Familie 65xx beziehen.



## 1.2 Realisierung in der Hardware

Da bei dem verwendeten Prozessor 6510 nur ein Adressraum von 64K zur Verfügung steht, ging man neue Wege, um alle oben genannten Features unterzubringen.

Das Zauberwort heißt Multiplexen.

Unter Multiplexen soll hier die zeitlich verschachtelte Benutzung derselben Leitungen von mehreren Systemkomponenten verstanden werden.

Es wird nämlich einfach davon ausgegangen, daß nicht alle Einrichtungen der Hardware zur gleichen Zeit benutzt werden. Das hört sich zwar wie eine Binsenweisheit an, jedoch besteht die Kunst darin, die Komponenten derart zu verknüpfen, daß alle zu einem logischen Block (z.B. I/O-Block, Videoblock usw.) gehörenden Einheiten ohne Umschalten zu erreichen sind.

Schon hier sei verraten, daß das Herzstück dieser komplexen Funktion ein spezielles IC ist, welches wir in Ermangelung eines anderen Namens einfach Adressraum-Manager nennen wollen. Mehr hierüber jedoch erst im Abschnitt 1.4.

Ein Beispiel für die Überlagerung von Komponenten, die denselben Adressbereich belegen:

Zeichengenerator und I/O-Bereich belegen dieselben Adressen. Dennoch kommt es nicht zum Konflikt, da der VIC beim Zugriff auf den Charactergenerator geschickt die Systemtaktlücken ausnutzt, in denen der Prozessor ohnehin nicht auf den Adressbus zugreift.

Bei soviel Überlappung können natürlich auch Konfliktfälle auftreten, die aber auf jeden Fall durch eine geeignete Software vermieden werden können.

Deshalb sollten Maschinenprogrammierer dieses Kapitel aufmerksam aufnehmen, da hier auf solche Reibungspunkte hingewiesen wird.

Das gesamte ROM und Teile des RAM überlappen sich über einen Bereich von 20K. Wie dieser Fall gelöst wird, ist im Abschnitt 1.4 nachzulesen.

Bleibt noch von einer weiteren Überlagerung zu berichten, die allerdings nicht den Adressbus, sondern den Peripheriebereich betrifft.

Dort sind einige I/O-Lines der CIAs doppelt belegt, nämlich sowohl mit der Tastatur als auch mit den Controlports außen am Rechner. Wie diese Klippe umschifft werden kann, ist im Abschnitt 4.7 ausführlich behandelt.

Es ist selbstverständlich, daß in einem derart komplexen Aufbau auch die Einzelkomponenten Aufgaben übernehmen müssen, die weit über ihre ihnen gewöhnlich zugedachten Funktionen hinausgehen.

Ein Paradebeispiel hierfür ist der Videocontroller 6567 (VIC):

Der Ram besteht aus dynamischen ICs 4164, organisiert zu 64K mal 1 Bit.

Dennoch sind in der Hardware keinerlei Komponenten enthalten, die die ziemlich komplizierte (im Vergleich mit statischen Rams) Adressierung vornehmen oder für den pünktlichen Refresh (Wiederauffrischen des Speicherinhaltes in kurzen Abständen) sorgen. Diese ganzen Angelegenheiten werden vom Videocontroller automatisch erledigt, neben seiner eigentlichen Aufgabe, für ein anständiges Bild zu sorgen.

### 1.3 Der Prozessor 6510 und seine Besonderheiten

Der CBM 64 hat als Prozessor die MPU 6510 (MPU=Micro Processing Unit).

Dies ist ein 8-Bit-Prozessor, der im CBM 64 mit einer Taktfrequenz von ca. 980KHz betrieben wird.

Die MPU 6510 ist ein neues IC aus der Familie 65xx. Der wesentliche Unterschied zum bekannten 6502 besteht darin, daß der Chip über sechs I/O-Leitungen verfügt.

Das hat in Minimalkonfigurationen gegenüber dem 6502, der über keinerlei I/O-Leitungen verfügt, den Vorteil, daß man ohne zusätzliche Peripherie-ICs über einzeln programmierbare Leitungen verfügt, sei es zur Verbindung mit der Außenwelt (z.B. Tastatur), sei es für interne Steuerzwecke.

Im CBM 64 unterstützen diese Leitungen zum einen den Kassettenbetrieb, zum anderen die Speicherverwaltung.

Der Befehlssatz ist der gleiche wie beim 6502.

Interessant für den Programmierer dürfte noch sein, daß die I/O-Leitungen die Adressen 0 und 1 belegen, nämlich 0 als Datenrichtungsregister und 1 als Datenregister.

Der Stack kann sich im Bereich \$0100 bis \$01FF bewegen.

Unter Stack versteht man einen Stapelspeicher, auf dem, z.B. beim Sprung in Unterprogramme, die Prozessorregister abgelegt werden können, um sie beim Rücksprung ins Hauptprogramm wieder hervorzuholen und mit den ursprünglichen Inhalten weiterzuarbeiten.

Zu der nun folgenden Pin-Belegung noch eine Anmerkung:

Es ist uns nicht gelungen, ein dem tatsächlichen Stand entsprechendes Datenblatt aufzutreiben. Wir haben zwar mehrere, doch auf jedem ist das Pinout anders dargestellt. Deshalb sind wir den empirischen Weg gegangen und haben kontrolliert, wo die einzelnen Leitungen denn nun tatsächlich hingehen. Hier nun das Ergebnis:

- 1     BIN Systemtakt Eingang; im CBM 64 ca. 980KHz.
- 2     RDY Ready; 0=Prozessor hält beim nächsten Lesesyklus an, bis RDY=1. Von dieser Möglichkeit macht man z.B. beim Betrieb langsamer Speicher Gebrauch.
- 3     -IRQ Interrupt Request; 0=Prozessor holt sich die nächste Befehlsadresse von \$FFFE und macht dort weiter. Dieser Umstand tritt nur ein, wenn der Interrupt erlaubt war (Bit 2 im Flag-Register=0).
- 4     -NMI Non-Maskable Interrupt; 0=Prozessor holt sich die nächste Befehlsadresse von \$FFFA und macht dort weiter.
- 5     AEC Address Enable Control; 0=Prozessor bringt Daten-Adress- und Steuerbus in den hochohmigen Zustand (Tri-State). Der Bus kann nun von anderen Einheiten betrieben werden, z.B. ein zweiter Prozessor.
- 6     VCC Betriebsspannung +5V
- 7 -20 A0-A13; Adressbus
- 21    GND
- 22-23 A14-A15; Adressbus
- 24-29 P5-P0; I/O-Pins
- 30-37 D7-D0; Datenbus
- 38    R/-W; 0=Schreibzugriff, 1=Lesezugriff  
Alle Zugriffe finden nur während RDY=1 statt.

- 39    02OUT; Systemtakt Ausgang zur Versorgung anderer Bausteine.
- 40    RES Reset; 0=Prozessor geht in den Ruhezustand. Beim Übergang von 0 nach 1 holt sich der Prozessor eine Adresse von \$FFFC und beginnt dort mit dem Programm.

#### 1.4 Speicherbelegungspläne

Aufgrund der mehrfachen Bereichsüberlappungen und der damit notwendigen Verwaltung des Adressraumes ergeben sich etliche mögliche Belegungen.

Dreh- und Angelpunkt des gesamten Komplexes ist ein IC, welches im Folgenden kurz Adress-Manager (AM) genannt wird. Beim AM handelt es sich um ein IC von der Sorte FPLA (Field Programmable Logic Array).

Derartige Bausteine enthalten eine Vielzahl potentieller logischer Verknüpfungen, die vom Anwender in der gewünschten Kombination programmiert werden können.

Im Falle des CBM 64 ist dies ein IC mit 16 Eingangsleitungen und 8 Ausgangsleitungen.

Durch die Programmierung wurde jeder Eingangskombination (eine aus 65536 möglichen) eine Ausgangskombination (eine aus 256 möglichen) zugeordnet.

Da dieses IC ein wesentlicher Bestandteil des CBM 64 ist, sollen hier die Ein- und Ausgangssignale erklärt werden mit Zweck (Ausgänge) und Herkunft (Eingänge).

Zunächst die Eingänge:

- CAS    Signal vom VIC zur Steuerung der Ramadressen.
- LORAM    Signal vom Prozessorport Bit 0
- HIRAM    wie oben, jedoch Bit 1
- CHAREN    wie oben, jedoch Bit 2. Soll das Auslesen des Zeichengenerators ermöglichen. Versuchen Sie das nie in Basic, da die vom Timer angestoßene Interruptroutine statt des erwarteten Timers dort nun den Charactergenerator vorfindet (Charactergenerator und Timer in der CIA belegen dieselben Adressen) und deshalb falsch verzweigt, was zu einem 'Absturz' des Betriebssystems führt.
- VA14    kommt von CIA 2 Port A Bit 0; stellt einen Teil der Basisadresse von Zeichengenerator und Videoram dar.
- A13-A15    Adressbussignale; dienen hier zur Dekodierung des I/O-Bereiches.
- BA    Signal vom VIC; wirkt auf die RDY-Leitung des Prozessors
- AEC    0=Prozessor steuert den Bus, 1=VIC steuert den Bus. Das Signal ist durch Invertierung von AEC des VIC gewonnen.
- R/-W    Steuersignal des Prozessors
- EXROM    Eingangssignal vom Expansion-Blot
- GAME    wie oben
- VA12-13    Adressbus des VIC

Nun die Ausgänge:

-CASRAM 0=Ram übernimmt das höherwertige Byte der Speicheradresse  
-BASIC 0=Basic-Rom ist selektiert  
-KERNAL 0=Betriebssystem-Rom ist selektiert  
-CHAROM 0=Auswahl des Charactergenerators  
GR/-W 0=Schreibzugriff auf das Farbram  
-I/O 0=I/O-Dekoder selektiert  
-ROML Signal zum Expansion-Slot  
-ROMH wie oben

Auf der nächsten Seite finden Sie eine (fast) vollständige Liste der Zuordnungen von Eingangskombinationen zu Ausgängen.

Zu dem 'fast' eine kleine Anmerkung:

Es wäre uns sicher ein Leichtes gewesen, dieses Buch mit allen der zu den 65536 möglichen Eingangskombinationen gehörigen Ausgangskombinationen zu füllen.

Bei ICs dieser Gattung gibt es jedoch eine große Menge redundanter Eingangsbits, d.h. Bits, bei denen sich in einer bestimmten Konstellation nichts am Ausgang ändert. Solche Bits sind in der Tabelle durch ein X gekennzeichnet.

Um diese Bits aber herauszufinden, haben wir ein Programm entwickelt, welches über vier Wochen ohne Unterbrechung lief und immer noch nicht ganz fertig war.

Deshalb mögen am Ende der Tabelle vielleicht einige Kombinationen fehlen. Das sind ein paar der Ausgangskombination \$FE und alle der Kombination \$FF. Die letztere ist ohnehin nicht von großer Bedeutung, da ja, wie Sie sehen, alle Ausgänge 0-aktiv sein müssen, um etwas zu bewirken.

Wenn Sie sich nicht durch diese Tabelle durchkämpfen wollen, finden Sie auf den nächsten Seiten bildlich in geraffter Form die wichtigsten Auswirkungen des AM auf die Speicherzuordnung.



[illegible]





#E000  
#DC00  
#D800  
#D000

SR RAM  
CIA 1/2 + I/O 0-1  
FARB-RAM  
VIC II / SID

#8000

#0000

In dieser Konfiguration könnte der Speicher von einem externen Prozessor betrieben werden, wobei die Peripheriebausteine mitbenutzt werden.

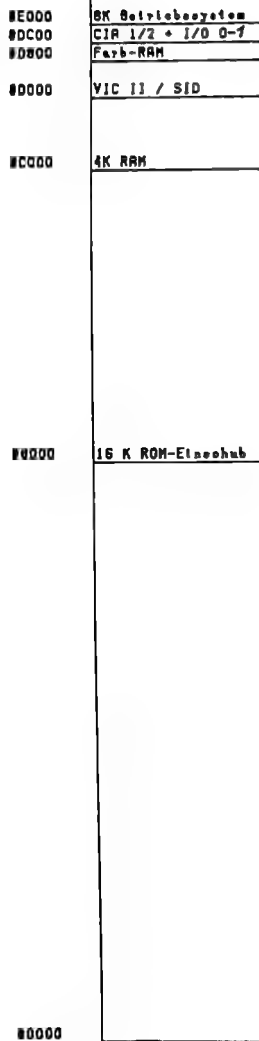
-LORAM =1  
-HIRAM =0  
-EXROM =X  
-GAME =1  
-CHAROM=1

10000

10000

Diese Form gleicht der  
vorigen, jedoch kann hier der  
I/O-Bereich nicht benutzt  
werden.

-LORAM =0  
-HIRAM =0  
-EXROM =1  
-GAME =1



Dies ist eine typische Konfiguration, wenn eine andere Sprache als Basic benutzt werden soll. Die andere Sprache (z.B. PASCAL) steckt in einer externen ROM-Kassette.

- LORAM =1
- HIRAM =1
- EXROM =0
- GAME =0
- CHAREN=1



8E000	Betriebssystem
8DC00	CIA 1/2 + I/O 0-1
8D800	Farb-RAM
8D500	VIC II / SID
8C000	4K RAM
8A000	8K BASIC
8B000	BASIC-Erweiterung
80000	

In dieser Konfiguration ist das Basic durch eine ROM-Kassette erweitert (z.B. TOOLKIT).

-LORAM =1  
 -HIRAM =1  
 -EXROM =0  
 -GAME =1  
 -CHAREN=1



#DF00	I/O 1
#DE00	I/O 0
#DD00	CIA 2
#DC00	CIA 1
#D800	Farb-RAM
#D400	SID-Chip
#D000	VIDEO-Centr.

Hier finden Sie die  
Aufschlüsselung des I/O-  
Bereiches, wenn -CHAREN=1.

Bei -CHAREN=0 belegt der  
Charactergenerator den ge-  
samten Bereich.

## 1.5 Die Erweiterungsschnittstelle

Hinten am CBM 64 befindet sich ein 44-poliger Steckplatz. Dieser sogenannte Expansion-Slot bietet vielfältige Erweiterungsmöglichkeiten.

An dieser Schnittstelle steht der gesamte Systembus mit den zugehörigen Steuerleitungen zur Verfügung.

An dieser Stelle werden z.B. Basic- und Betriebssystemerweiterungen (Toolkit, Maschinensprachemonitor usw.), Spiele und Ein- Ausgabeerweiterungen (IEC-Bus) angeschlossen.

Es ist sogar möglich, mit einem hier angeschlossenen Prozessor auf die Komponenten im Inneren des Gerätes zuzugreifen.

Zur Benutzung von mehreren Erweiterungen, z.B. Programmierhilfe und Maschinensprache, läßt sich diese Schnittstelle durch geeignete Vorrichtungen auch vervielfachen.

Module, die für den VC-20 vorgesehen sind, passen weder logisch noch physikalisch an diese Schnittstelle.

Wollen Sie es dennoch versuchen, so müssen die vom VC-20 bekannten Blockauswahlsignale durch Dekodierung der Adressbits 13-15 gewonnen werden.

Da viele Signale dabei sind, die nicht selbsterklärend sind, hier die Pinbelegung:

1	GND
2 -3	+5V
4	-IRQ; mit IRQ des Prozessors verbunden
5	CR/-W; mit R/-W des Prozessors verbunden
6	DOT CLOCK; Punktrastertakt für den VIC, ca. 7,83 MHz
7	-I/OI; gewöhnlich =0 im Adressbereich \$DE00 bis \$DEFF
8	-GAME; Eingang zum AM; Auswirkung siehe 1.4
9	-EXROM; wie oben
10	-I/O2; gewöhnlich =0 im Bereich \$DF00 bis \$DFFF
11	-ROML; Ausgang vom AM. Siehe 1.4
12	BA; Signal vom VIC, welches die Gültigkeit von Lesedaten anzeigt
13	-DMA; Eingang. 0=Busssystem für den externen Zugriff reservieren.
14-21	CD7-CD0; Datenbus
22	GND
A	GND
B	-ROMH; Ausgang vom AM. Siehe 1.4
C	-RESET
D	-NMI
E	02; Systemtakt Ausgang
F -Y	CA15-CA0; Adressbus
Z	GND

Wesentliche Unterschiede zum Expansionslot des VC-20 sind:  
Anderer Abstand der Steckkontakte untereinander; 2,54mm hier, 3,96mm beim VC-20.

Die Block- und Ramauswahlsignale -RAM1-3 und -BLK1-5 fehlen völlig, da der Adressbus vollständig herausgeführt ist. Beim VC-20 sind es nur die Adressbits 0-12.

Anderer Umfang der I/O-Bereiche; hier 256 Byte, beim VC-20 1K.

Wenn Sie diesen Abschnitt im Zusammenhang mit dem Vorigen sehen, dürfte klar werden, welchen Einfluß man selbst von außen auf die Speicherkonfiguration hat. Das gilt insbesondere für die Signale -EXROM und -GAME, da hiermit die Speicherzuordnung derart umkonfiguriert werden kann, daß nach dem Einschalten nicht einmal mehr das erwartete 'READY' auf dem Bildschirm erscheint. Daher sind die entsprechenden Signale nur mit Vorsicht zu genießen.

## 1.6 Der User-Port

Mit dem User-Port bietet der CBM 64 eine Schnittstelle, die zur Steuerung, der Name sagt es, anwenderspezifischer Peripheriegeräte dient. Das wären im einfachsten Falle über Treibertransistoren angeschlossene Lampen, das könnte aber auch ein Drucker mit 8-Bit-Parallelschnittstelle (Centronics) sein, den Sie vielleicht zufällig besitzen und gerne am CBM 64 betreiben würden.

Ein geeigneter Stecker zum Anschluß an den Userport ist z.B. der Cardcon-Stecker der Firma TRW mit der Nummer 251-12-50-170 aus der Serie Cinch-Connectors. Es handelt sich hierbei um einen zweireihigen Platinindirektstecker mit 2x12 Kontakten im Abstand von 3,96mm.

Der Userport besteht im wesentlichen aus einem 8-Bit-Port und diversen Steuerleitungen, die im Folgenden näher vorgestellt werden:

- 1 GND
- 2 +5V; mit max. 100mA belastbar
- 3 -RESET; mit der gleichnamigen Prozessorleitung verbunden.
- 4 CNT1; verbunden mit CNT von CIA 1
- 5 SP1; mit SP von CIA 1 verbunden
- 6 CNT2; Leitung CNT von CIA 2
- 7 SP2; verbunden mit SP von CIA 2
- 8 -PC2; Handshake-Ausgang von CIA 2
- 9 ATN OUT; Steuerleitung des seriellen IEC-Bus, stammt von PA3 der CIA 2.
- 10 9V AC; Wechselspannung; mit max. 100 mA belastbar.
- 11 Gegenpol zu oben
- 12 GND
- A GND
- B -FLAG2; Handshake-Eingang von CIA 2
- C-L PB0-PB7; I/O-Lines von CIA 2
- M PA2; I/O-Line von CIA 2. Diese Leitung ersetzt den von den anderen CBMs bekannten CB2 der VIA 6522.
- N GND

Einige der oben aufgeführten Leitungen haben im CBM 64 bereits fest zugeordnete Funktionen. Hierzu und auch zur Handhabung der CIAs schauen Sie bitte im Kapitel 4 nach.

Wollen Sie an den Userport Geräte anschließen, die eigentlich zum Anschluß an den Userport des VC-20 oder CBM 8032 o.ä. vorgesehen sind, so ist das prinzipiell nur bei den Geräten möglich, die zu ihrem Betrieb ausschließlich die Pins A-N, 1 und 12 benötigen. Auf jeden Fall müssen Sie die programmtechnisch unterschiedliche Handhabung der Pins B und M beachten. Hierzu gibt das Kapitel 4 weitere Auskunft.

#### 1.6.1 Die Programmierung des User-Port in Basic

Wenn Sie sich zu denjenigen zählen, die Basic und alles, was dazugehört wie ihre Westentasche kennen, denen jedoch die nicht standardmäßigen Anschlüsse (in die nicht einfach irgendein Peripheriegerät eingesteckt werden kann) verdächtig sind und als nur Insidern vorbehaltenes Relikt aus den Anfängen der 'Computerei' erscheint, dann ist Ihnen dieser Abschnitt gewidmet.

Wir wollen Ihnen zeigen, wie Sie eine einfach aufzubauende Schaltung am Userport anschließen können und diese im vertrauten Basic handhaben können. Die Anordnung besteht lediglich aus vier Schaltern, vier Leuchtdioden, acht Widerständen und einem IC. Sie werden damit die Grundbegriffe der Datenein- und ausgabe über den Userport sehr leicht verstehen und sich die so gewonnenen Erfahrungen für eigene Projekte zunutze machen können. Die Schaltung finden Sie am Ende dieses Abschnitts. Sie ist Ihrer Einfachheit wegen nicht weiter kommentiert.

Wegen der Vielzahl der belegten Anschlüsse des Userport (es sind mehr als z.B. bei den Rechnern der CBM-Reihe) muß zunächst geklärt werden, was am Userport tatsächlich 'User-' ist.

Falls Sie nicht zufällig eine RS-232-Cartridge betreiben, können Sie die folgenden Anschlüsse ohne Rückwirkung auf die übrigen Funktionen des Rechners benutzen:

1-2, 4-8, 10-12, A-N

Die Bedeutung dieser Pins entnehmen Sie bitte dem vorangegangenen Abschnitt.

Nun zu unserem Beispiel:

Die Datenleitungen PB0-PB7 lassen sich individuell auf Eingabe oder Ausgabe programmieren. Um dies zu demonstrieren, benutzen wir in unserem Beispiel die Leitungen PB0-PB3 als Eingabe, und die Leitungen PB4-PB7 als Ausgabe. Die Festlegung der Datenrichtung geschieht einfach durch das Laden des Datenrichtungsregisters für den Datenport B auf der Adresse 56579. Ein gesetztes Bit (=1) bedeutet Ausgabe auf dem korrespondierenden Bit des Datenport B (Adresse 56577), ein rückgesetztes Bit (=0)

bedeutet Eingabe. Um für unser Beispiel die Datenrichtungen festzulegen, d.h. Bit 0-3 als Eingabe, 4-7 als Ausgabe, benutzen Sie einfach folgenden Befehl:

POKE 56579,240

Damit sind die oberen 4 Bits =1 gesetzt und somit stehen die korrespondierenden Datenbits des Port B auf Ausgabe, die übrigen auf Eingabe.

Wie wird nun unsere kleine Schaltung des weiteren gehandhabt?

Nichts einfacher als das:

PRINT PEEK(56577)AND15

signalisiert Ihnen den Zustand der vier Schalter, und mit

POKE 56577,X

können Sie die Leuchtdioden ein- oder ausschalten, wobei der Wert X nur aus den Zahlen 16,32,64,128 zusammengesetzt sein darf oder 0 ist, wenn alle Lämpchen aus sein sollen.

Sollten Sie eigene Projekte am Userport anschließen wollen, so beachten Sie bitte, um eine Beschädigung des Rechners zu vermeiden, unbedingt folgendes:

Bei Verwendung des Userport als Eingang darf die Eingangsspannung nur im Bereich 0-5 Volt liegen. Eine Spannung im Bereich 0-0,6 Volt wird beim Auslesen des Datenports als 0 interpretiert, eine solche von 1,6-5 Volt als 1. Der Bereich von 0,7 bis 1,5 Volt ist indifferent, d.h. er kann zufällig als 0 oder 1 erkannt werden.

Bei Verwendung als Ausgang beachten Sie bitte, daß die Ausgänge nur eine Belastung eines TTL-Eingangs aushalten. Sie könnten also keinesfalls eine Leuchtdiode direkt anschließen, was langfristig zur Zerstörung der CIA führen würde. Es empfiehlt sich immer eine Pufferstufe, wie auch in unserem Beispiel.

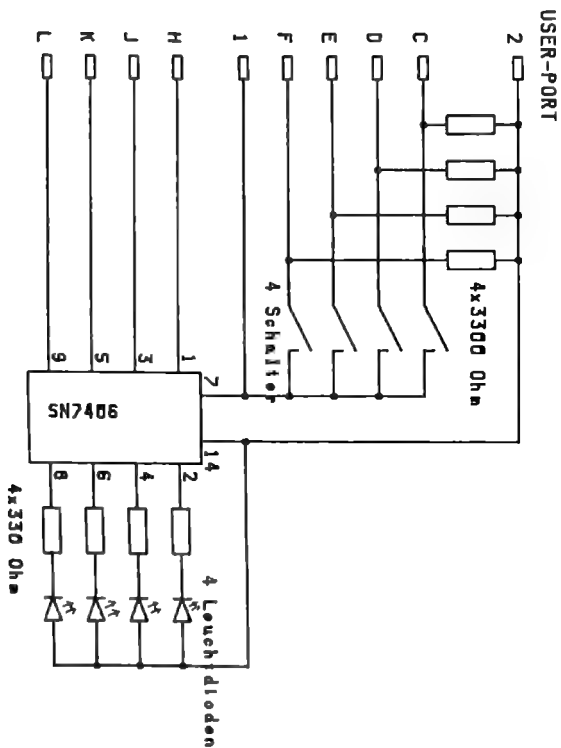
Vermeiden Sie unbedingt, ein auf Ausgabe programmiertes Portbit mit einer Fremdspannung von außen zu beaufschlagen, was zur unmittelbaren Zerstörung führt. Überlegen Sie sich daher gut, welchen Wert Sie in das Datenrichtungsregister laden, damit Sie nicht versehentlich ein für die Eingabe vorgesehenes Bit auf Ausgabe programmieren.

Wenn Sie die Stromversorgung für Ihr Projekt dem Userport entnehmen wollen, beachten Sie bitte, daß Sie die beiden zur Verfügung stehenden Spannungen nicht mit mehr als je 100mA belasten. Bei leichten Übertretungen wird zunächst der Kassettenrecorder streiken, danach verabschiedet sich die Sicherung im Innern des C64 und evtl. auch die Primärsicherung im Trafogehäuse. Zerstört wird dabei jedoch nichts weiter.

Dies sollte nur eine kleine Anleitung zur Bedienung des Userport in einem einfachen Anwendungsfall sein. Wollen Sie für komplexere Aufgaben auch die anderen Leitungen benutzen, so orientieren Sie sich bezüglich deren Handhabung bitte im

## Kapitel 4.

Grundsätzlich gilt dafür auch das oben Gesagte und selbstverständlich lassen sich diese Leitungen auch bequem von Basic aus 'POKEN' und 'PEEKEN'.



## Kapitel 2 : DER SYNTHESIZER UND SEINE PROGRAMMIERUNG

### 2.1 Der Sound-Controller 6581

#### 2.1.1 Allgemeines über den SID 6581

Mit dem im CBM 64 enthaltenen Synthesizer haben Sie die Möglichkeit, alle denkbaren Geräusche, vom Flötenton bis zur Dampflok, zu produzieren.

Während handelsübliche Synthesizer im allgemeinen über nur eine Stimme verfügen (monophon), haben Sie beim CBM 64 einen dreistimmigen (polyphonen) Synthesizer.

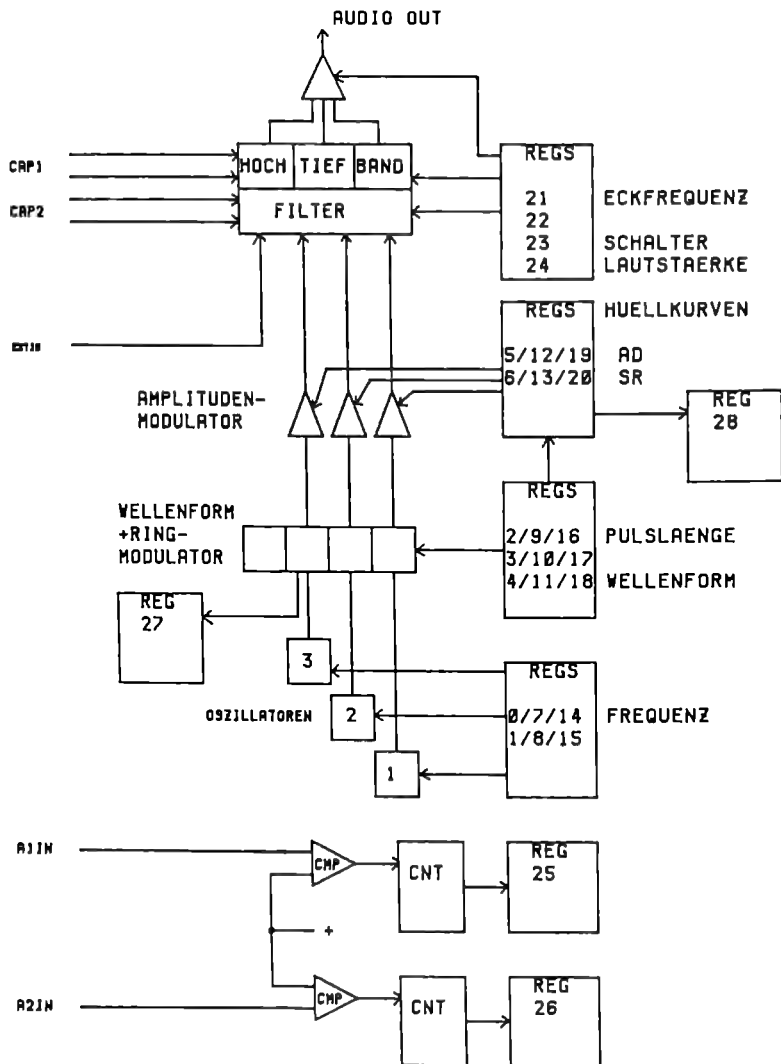
Alle der bei einem Synthesizer zur Klangerzeugung notwendigen Elemente befinden sich hier auf einem einzigen IC, nämlich dem Sound Interface Device (SID) 6581 ist ein neuer Peripheriebaustein aus der 65xx-Familie. Er verfügt über:

- \* 3 einzeln programmierbare, unabhängige Oszillatoren (Stimmen)
- \* 4 mischbare Schwingungsarten für jede Stimme
- \* 3 mischbare Filter (Hoch- Tief- Bandpass)
- \* Hüllkurvengenerator (ADSR-Control) für jede Stimme
- \* 2 kaskadierbare Ringmodulatoren
- \* Verfremdungsmöglichkeit für externe Signalquelle
- \* 2 8-Bit A/D-Wandler

Das Blockschema des SID finden Sie auf der nächsten Seite.

Pinbelegung des 28-poligen Gehäuses:

- 1 - 2 CAP1A,CAP1B; Anschluß des Kondensators für die programmierbaren Filter. Empfohlene Kapazität 220pF.
- 3 - 4 CAP2A,CAP2B; wie 1-2.
- 5 -RES (Reset); =0 bringt den SID in den Grundzustand.
- 6 02 (Systemtakt 2); alle Datenbusaktionen finden nur während 02=1 statt.
- 7 R/W (Read/-Write); 0=Schreibzugriff, 1=Lesezugriff.
- 8 -CS (Chip Select); 0=Datenbus gültig, 1=Datenbus hochohmig (Tri-State).
- 9 -13 A0-A4 (Adressbits 0-4); dienen zur Auswahl eines der 29 Register des SID.
- 14 GND (Masse); Achtung: Der SID sollte eine eigene Masseleitung zur Stromversorgung besitzen, um gegenseitige Beeinflussungen mit anderen Systemkomponenten zu vermeiden!
- 15-22 D0-D7 Datenleitungen zum/vom Prozessorsystem.
- 23 A2IN (Analog Input 2); Handhabung unbedingt unter 4.1.3 nachlesen!
- 24 A1IN Wie 23, allerdings für A/D-Wandler 1.
- 25 VCC Versorgungsspannung +5V.
- 26 EXT IN (External Input); Eingang für externe Audiosignale, die durch SID verfremdet werden sollen.
- 27 AUDIO OUT Summenausgang aller im SID erzeugten oder behandelten Signale.
- 28 VDD Versorgungsspannung +12V.





Der SID 6581 besteht aus drei Synthesizerstimmen, die sowohl unabhängig voneinander als auch in Verbindung miteinander (auch in Verbindung mit einer externen Audioquelle) benutzt werden können, um äußerst komplexe Schwingungsformen zu erzeugen.

Jede Stimme besteht aus einem Oszillator, einem Schwingungsformgenerator, der vier unterschiedliche Wellenformen erzeugen kann, einem Hüllkurvengenerator und einem Amplitudenmodulator.

Der Oszillator erzeugt die Grundfrequenz im Bereich 0-8200Hz (bei einem Systemtakt von ca. 1MHz) mit einer Auflösung von 16 Bit. Er kann vier Schwingungsformen hervorbringen, nämlich Dreieck, Sägezahn, Rechteck (mit variablem Puls-Pauseverhältnis) und rosa Rauschen.

Jede dieser Schwingungsformen hat den ihr eigentümlichen Gehalt an Oberwellen, sodaß schon allein durch die Wahl der Wellenform die Klangfarbe beeinflusst werden kann.

Eine Dreieckschwingung hat, wegen ihrer Verwandtschaft zum Sinus (was den Obertongehalt angeht), einen sehr weichen Klang, etwa wie eine Holzflöte.

Die Sägezahnschwingung enthält ein vollständiges Obertonspektrum in gleichmäßiger Verteilung. Sie klingt daher scharf, wie z.B. eine Trompete.

Der Rechteckschwingung fehlen bestimmte Obertonbereiche. Sie klingt etwas hohl, einer Klarinette sehr ähnlich.

Das rosa Rauschen besteht aus einer zufälligen Aufeinanderfolge verschiedener Frequenzen innerhalb festgelegter Grenzen.

Der Verlauf der Lautstärke über einen einzelnen Ton wird durch einen Amplitudenmodulator beeinflusst, der vom Hüllkurvengenerator gesteuert wird. Wenn dieser durch ein Triqgerbit (vergleichbar mit einer Taste am Piano) angestoßen wird, beginnt der programmierte Verlauf des Tones von Anklingphase über Haltephase zur Ausklingphase.

Die Ausgänge aller Stimmen können außerdem noch über programmierbare Filter geschickt werden, um die Klangfarbe in weiten Grenzen zu beeinflussen.

Um das Maß voll zu machen, können die Stimmen 1 und 2 noch von der Stimme 3 ringmoduliert werden, d.h. es entstehen außer der Grundstimme noch Summe und Differenz mit der Stimme 3.

Soll während des Ablaufs eines einzelnen Tones noch dessen Klangfarbe verändert werden (z.B. WAH-WAH-Effekt), so kann dies bei der Stimme 3 dadurch bewirkt werden, daß man den Augenblickswert des Hüllkurvengenerators ausliest und in Abhängigkeit von dessen Stand die Filterfrequenz oder auch die Oszillatorfrequenz verändert. Außerdem kann noch der gegenwärtige Stand des Rauschgenerators von Stimme 3 ausgelesen werden. Man hat damit einen ausgezeichneten Zufallszahlengenerator.

Der SID enthält auch noch zwei 8-Bit A/D-Wandler. Damit kann der jeweilige relative Wert zweier am Controlport anzuschließender Potentiometer (z.B. Paddles) abgefragt werden. Die häufigste Anwendung hierfür werden sicher Spiele bieten, jedoch sind auch andere Verwendungen denkbar wie z.B. die Steuerung von Tonhöhe und Lautstärke (oder anderer Parameter) des Synthesizers, ja sogar, bei Anschluß von Kaltleitern, die Verwendung als Thermometer.

## 2.1.2 Registerbeschreibung des SID

Die Basisadresse des SID im CBM 64 ist \$D400(54272)

REG 0 Oszillatorfrequenz niederwertiges Byte für Stimme 1

REG 1 Oszillatorfrequenz höherwertiges Byte für Stimme 1

REG 2 Pulsbreite niederwertiges Byte für Stimme 1

REG 3 Pulsbreite höherwertiges Byte für Stimme 1  
Die Register 2 und 3 bestimmen das Puls-  
Pauseverhältnis des Rechteckausgangs von Stimme 1.  
Von Register 3 werden nur die Bits 0-3 benutzt.

REG 4 Steuerregister für Stimme 1

Bit 0 KEY: Steuerbit für den Ablauf des  
Hüllkurvengenerators. Beim Übergang von 0 nach 1  
steigt die Lautstärke von Stimme 1 innerhalb der in  
REG 5 programmierten "Attack"-Zeit von null auf den  
Maximalpegel und fällt dann in der ebenfalls in REG 5  
eingestellten "Decay"-Zeit auf den in REG 6 gewählten  
"Sustain"-Pegel ab, auf dem sie bleibt, bis das  
Steuerbit wieder 0 wird. Jetzt geht die Lautstärke  
innerhalb der in REG 6 programmierten "Release"-Zeit  
auf null.

Bit 1 SYNC: 1=Oszillator 1 wird mit Oszillator 3  
synchronisiert. Dieses Bit hat auch Wirkung, wenn die  
Stimme 3 stummgeschaltet ist.

Bit 2 RING: 1=Der Dreieckschwingungsausgang von  
Oszillator 1 wird durch ein Frequenzgemisch (Summe  
und Differenz der Frequenzen von Oszillator 1 und 3)  
ersetzt. Dieser Effekt tritt auch dann ein, wenn  
Stimme 3 stummgeschaltet ist.

Bit 3 TEST: Wenn zusammen mit dem Rauschgenerator  
noch eine weitere Schwingungsform desselben  
Oszillators ausgewählt wurde, kann es vorkommen, daß  
der Rauschgenerator blockiert. Die Blockade kann  
durch dieses Bit wieder aufgehoben werden.

Bit 4 TRI: 1=Dreieckschwingung ausgewählt.

Bit 5 SAW: 1=Sägezahnsschwingung ausgewählt.

Bit 6 PUL: 1=Rechteckschwingung ausgewählt. Das Puls-  
Pauseverhältnis dieser Schwingung wird in REG 2 und  
REG 3 eingestellt.

Bit 7 NSE: 1=Rauschgenerator ausgewählt.

Anmerkung zu den Bits 4-7: Es ist praktisch möglich,  
mehrere Schwingungsformen gleichzeitig auszuwählen.  
Zu beachten ist jedoch, außer dem zu Bit 3 Gesagten,  
daß das Ergebnis nicht etwa die Summe aller Formen  
darstellt, sondern vielmehr eine logische  
UND-Verknüpfung der Komponenten.

REG 5 ATTACK/DECAY

Bit 0-3 Diese Bits bestimmen die Zeit, in der die  
Lautstärke vom Maximum auf den Sustainpegel abfällt.  
Der einstellbare Bereich beträgt 6msec bis 24sec.

Bit 4-7 Hiermit wird die Zeit festgelegt, in der die  
Lautstärke nach Setzen des KEY-Bits von null auf das  
Maximum ansteigt. Der einstellbare Bereich beträgt  
2msec bis 8sec.

- REG 6 SUSTAIN/RELEASE  
 Bit 0-3 Mit diesen Bits wird die Zeit eingestellt, innerhalb der die Lautstärke nach Rücksetzen des KEY-Bits vom Sustainpegel auf null abfällt. Der einstellbare Bereich beträgt 6msec bis 24sec.  
 Bit 4-7 Diese Bits geben den Sustainpegel an, d.h. die Lautstärke, mit der der Ton nach Ablauf der Decayzeit andauert.
- REG 7- Diese Register steuern die Stimme 2 analog zu den  
 REG 13 Registern 0-6 mit folgenden Ausnahmen:  
 SYNC synchronisiert Oszillator 2 mit Oszillator 1.  
 RING ersetzt den Dreiecksausgang von Oszillator 2 mit den ringmodulierten Frequenzen der Oszillatoren 2 und 1.
- REG 14- Diese Register steuern die Stimme 3 analog zu den  
 REG 20 Registern 0-6 mit folgenden Ausnahmen:  
 SYNC synchronisiert Oszillator 3 mit Oszillator 2.  
 RING ersetzt den Dreiecksausgang von Oszillator 3 mit dem Frequenzgemisch aus den Oszillatoren 3 und 2.
- REG 21 Filterfrequenz niederwertiges Byte  
 Es werden nur die Bits 0-2 benutzt.
- REG 22 Filterfrequenz höherwertiges Byte  
 Die 11-Bit-Zahl der Register 21 und 22 bestimmt die Filtereckfrequenz, bzw. -mittelfrequenz.  
 Im CBM 64 errechnet sich diese Frequenz folgendermaßen:  

$$F = (30 + W * 5,8) \text{ Hz}$$
 wobei W die 11-Bit-Zahl darstellt.
- REG 23 Filterresonanz und -schalter  
 Bit 0 1=Stimme 1 wird über den Filter geleitet.  
 Bit 1 1=Stimme 2 wird über den Filter geleitet.  
 Bit 2 1=Stimme 3 wird über den Filter geleitet.  
 Bit 3 1=Die externe Signalquelle wird gefiltert.  
 Bit 4-7 Diese Bits bestimmen die Resonanzfrequenz des Filters. Diese benutzt man dazu, bestimmte Ausschnitte des Frequenzspektrums hervorzuheben. Die Wirkung kann besonders gut bei der Sägezahnsschwingung beobachtet werden.
- REG 24 Dieses Register hat folgende Funktionen:  
 Bit 0-3 Gesamtlautstärke  
 Bit 4 Schaltet den Tiefpaßzweig des Filters ein.  
 Bit 5 Schaltet den Bandpaßzweig des Filters ein.  
 Bit 6 Schaltet den Hochpaßzweig des Filters ein.  
 Hoch- und Tiefpaßfilter haben eine Flankensteilheit von 12dB/Oktave. Der Bandpaßfilter hat eine solche von 6dB/Oktave.  
 Es kann mehr als ein Filter eingeschaltet sein. Ist z.B. Hoch- und Tiefpaß eingeschaltet, resultiert daraus eine Bandsperre.

Um den Einfluß der Filter zu Gehör zu bringen, muß wenigstens ein Filter eingeschaltet sein und wenigstens eine Stimme über den Filter geleitet werden.

Allgemein gesehen wird das Filter dazu benutzt, bestimmte Bereiche aus einem Frequenzspektrum herauszufiltern.

Daher erlaubt die Filterung eine viel feinfühligere und ausgeklügeltere Beeinflussung des Klangbildes, als es durch bloße Auswahl der Schwingungsform möglich wäre.

Verändert man die Filterfrequenz während des Ablaufs eines Tones (bei kurzen Tönen nur in Maschinensprache möglich), so lassen sich die verschiedensten Instrumente perfekt nachbilden.

Bit 7 1=Stimme 3 unhörbar. Von dieser Möglichkeit kann man Gebrauch machen, wenn der Verlauf der Stimme 3 nur zur Parametergewinnung für die anderen Stimmen dienen soll (siehe hierzu Register 27 und 28).

Auf alle zuvor aufgeführten Register kann nur ein Schreibzugriff durchgeführt werden. Ein Lesezugriff bringt keine Aussage.

Alle folgenden Register können nur gelesen werden.

REG 25 A/D-Wandler 1

REG 26 A/D-Wandler 2

REG 27 Rauschgenerator 3

Dieses Register liefert eine Zufallszahl, die dem augenblicklichen Stand des Rauschgenerators 3 entspricht. Der Generator muß hierzu eingeschaltet sein, jedoch kann die Stimme 3 unhörbar sein (Bit 7 in REG 24 =1).

REG 28 Hüllkurvengenerator 3

Aus diesem Register kann man den augenblicklichen Stand der relativen Lautstärke von Stimme 3 entnehmen. So könnte man entsprechend dem Lautstärkeverlauf die Frequenz oder die Filterparameter ändern. Ein Beispiel hierzu finden Sie im Abschnitt 2.2.

### 2.1.3 Der Analog/Digitalwandler

Ein A/D-Wandler ist eine Einrichtung zur Umwandlung eines analogen Signals (z.B. Spannung) in einen digitalen Wert.

Die prinzipielle Schwierigkeit bei einer solchen Umwandlung besteht darin, einen analogen Wert mit unendlich feiner Abstufung in einen digitalen Wert mit endlicher Abstufung (feste Intervalle) umzuformen.

Es entsteht dabei zwangsläufig ein großer Fehler von +/- einem kleinsten digitalen Schritt.

Der SID 6581 enthält zwei A/D-Wandler. Hierbei handelt es sich um eine Anordnung mit einer intern erzeugten Referenzspannung von ca. 2.5V.

Der Meßvorgang besteht darin, daß eine externe Kapazität zunächst entladen wird und anschließend ein Wert in Register 25 bzw. 26 übernommen wird, der der benötigten Zeit für eine erneute Aufladung der Kapazität auf die Referenzspannung entspricht. Dieser Vorgang wiederholt sich zyklisch.

### 2.1.3.1 Die Handhabung des A/D-Wandlers

Aus dem oben Gesagten ergibt sich, daß nur eine potentiometrische Beschaltung des Wandlers in Frage kommt. Als Meßwertaufnehmer eignen sich demnach nur veränderliche Widerstände in irgendeiner Form, z.B. Photowiderstände, Heißleiter, Kaltleiter usw.

Sollen Spannungen gemessen werden, so müssen diese zuvor in eine geeignete Form umgewandelt werden, z.B. mit Hilfe eines Unijunction-Transistors.

Die Meßanordnung sieht einfach so aus, daß an das eine Ende des Meßwiderstandes +5V angelegt werden (an den Controlports des CBM 64 verfügbar) und das andere Ende mit dem Analogeingang des SID (ebenfalls an den Controlports verfügbar, Bezeichnung POTX und POTY) verbunden wird. Der aus den Registern 25 und 26 ausgelesene Wert ist ein Maß für den Widerstand.

Um die ganze Skala von 8 Bit ausnutzen zu können, muß sich der Widerstand im Bereich von 200 Ohm (nicht kleiner!!!) bis 200 Kiloohm bewegen.

Die programmtechnische Handhabung der A/D-Wandler wird im nächsten Abschnitt behandelt.

### 2.1.3.2 Die Verwendung von Paddles

An den CBM 64 können handelsübliche Paddles angeschlossen werden.

Sie werden einfach in die Controlports 1 und/oder auf der rechten Seite des Gerätes eingesteckt.

Hinter einem Paddle verbirgt sich nichts weiter als ein Potentiometer, welches auf die im vorigen Abschnitt erläuterte Weise mit dem SID verbunden ist, und eine Taste, welche auf die Joystickposition LINKS für das eine Paddle und RECHTS für das andere Paddle wirkt.

Problematischer ist das Verfahren, die Paddlewerte programmtechnisch auszuwerten, da, wie im Abschnitt 8.7 näher erklärt, sich einige Bits zur Paddlesteuerung und -abfrage die CIAs 1 und 2 mit der Tastatur teilen müssen.

Zum einen sind dies die Tasten an den Paddles. Zum anderen haben wir die Anschlußmöglichkeit für zwei Paddlepaare, also vier Paddles. Da der SID aber nur über zwei A/D-Wandler verfügt, sind die Paddles über einen Analogschalter geführt. Um diesen zu bedienen, werden zwei weitere Bits der CIA 1

herangezogen.

Es muß also auch hier zur Auswertung der A/D-Wandler die Tastatur lahmgelegt werden, allerdings nur für die Zeit des tatsächlichen Zugriffs, da man sonst mit der Tastatur nicht mehr arbeiten könnte.

Die Lösung bringt folgendes kleine Maschinenprogramm, welches, als DATA-Statement in ein Basicprogramm eingebunden, wie weiter unten gezeigt, den komfortablen Zugriff auf alle Parameter von vier angeschlossenen Paddles erlaubt.

Das Programm ist in einem Bereich angelegt, der vom Betriebssystem nicht benutzt wird.

Die Rückmeldungen belegen einige Bytes, die nur während einer Kassettenoperation anderweitig belegt sind.

```
CFBE SEI      :Tastaturabfrage verhindern
CFBF LDA #$80 :Parameter für Paddlesatz A
CFC1 JSR $CFEC :A/D-Werte A1 und A2 holen
CFC4 STX $033C :und sicherstellen
CFC7 STY $033D
CFCA LDA $DC00 :Tasten A aus CIA 1 holen
CFCD AND #$0C  :benötigte Bits filtern
CFCF STA $029F :und sicherstellen
CFD2 LDA #$40  :Parameter für Paddlesatz B
CFD4 JSR $CFEC :A/D-Werte B1 und B2 holen
CFD7 STX $033E :und sicherstellen
CFDA STY $033F
CFDD LDA $DC01 :Tasten B aus CIA 2 holen
CFE0 AND #$0C  :benötigte Bits filtern
CFE2 STA $02A0 :und sicherstellen
CFE5 LDA #$FF  :alle Bits Ausgang in CIA 1
CFE7 STA $DC02 :um Tastaturabfrage wieder
CFEA CLI      :zu erlauben
CFEB RTS      :Rückkehr ins Basicprogramm
CFEC STA $DC00 :Paddlesatz auswählen
CFEF ORA #$C0  :und entsprechende Bits
CFF2 STA $DC02 :auf Ausgang setzen
CFF4 LDX #$0    :Verzögerungsschleife zur
CFF6 DEX        :Beruhigung des
CFF7 BNE $CFF6 :A/D-Eingangs
CFF9 LDX $D419 :A/D 1 holen
CFFC LDY $D41A :A/D 2 holen
CFFF RTS      :Rückkehr ins Hauptprogramm
```

Hier nun das Basicprogramm. Schließen Sie die Paddles an, laden und starten Sie das Programm und schauen Sie, was geschieht.

```
10 DATA 120,169,128,32,236,207,142,60,3,140,61,3,173
20 DATA 0,220,41,12,141,159,2,169,64,32,236,207,142
30 DATA 62,3,140,63,3,173,1,220,41,12,141,160,2,169
40 DATA 255,141,2,220,88,96,141,0,220,9,192,141,2
50 DATA 220,162,0,202,208,253,174,25,212,172,26,212,96
60 FOR M=53182 TO 53247
70 READ A: POKE M,A: NEXT: REM Maschinenprogramm einlesen
80 AX=830: REM Paddle 1 am Controlport 1
```

```

90 AY=831: REM Paddle 2 am Controlport 1
100 BA=672: REM Tasten von Paddlepaar A
110 BX=828: REM Paddle 1 am Controlport 2
120 BY=829: REM Paddle 2 am Controlport 2
130 BB=830: REM Tasten von Paddlepaar B
140 SYS 53182: REM Alle Werte holen
150 PRINT PEEK(AX)" "PEEK(AY)" "PEEK(BA)" ";
160 PRINT PEEK(BX)" "PEEK(BY)" "PEEK(BB)
170 GOTO 140

```

## 2.2 Die Programmierung des SID 6581

Dieser Abschnitt beschränkt sich auf die Programmierung des Synthesizers, da die Handhabung der A/D-Wandler schon in den Abschnitten 2.1.3.1 und 2.1.3.2 erklärt wurde.

Zur einfachen Erzeugung eines Tones sollten Sie zunächst grundsätzlich so vorgehen, daß Sie zuerst Register 24 laden, dann die ADSR-Register (5,6,12,13,19,20), anschließend die übrigen Register, am besten in der Reihenfolge ihrer Stimmenzugehörigkeit, mit Ausnahme der Register 4,11 und 18. Diese laden Sie zum Schluß (Bit 0 nicht vergessen!), und schon erklingt der Ton.

Folgendes kleine Programm soll Sie mit den Schwingungsformen und dem Tonumfang des Synthesizers bekannt machen:

```

10 S1=54272: REM Stimme 1
20 S2=54279: REM Stimme 2
30 S3=54286: REM Stimme 3
40 FL=54293: REM Filter Lo-Byte
50 FH=54294: REM Filter Hi-Byte
60 RS=54295: REM Resonanz + Schalter
70 PL=54296: REM Passart + Lautstärke
80 POKE S1+4,0: POKE S2+4,0: POKE S3+4,0
100 POKE S1+2,0: POKE S1+3,0
110 POKE S1+5,0: POKE S1+6,240
120 POKE RS,0: POKE PL,15
130 PRINT "DREIECK"
140 T=16: GOSUB 300
150 PRINT "SÄGEZAHN"
160 T=32: GOSUB 300
170 PRINT "RECHTECK"
180 T=64: GOSUB 300
190 PRINT "RAUSCHEN"
200 T=128: GOSUB 300
210 END
300 POKE S1,0: POKE S1+1,0
310 POKE S1+4,T+1: REM Ton einschalten
320 FOR I=0 TO 255: FOR J=0 TO 255 STEP 30
330 POKE S1,J: POKE S1+1,I
340 NEXT J,I
350 POKE S1+4,T: REM Ton ausschalten
360 RETURN

```

Die Zeilen 10 bis 80 sollten vor jedem Ihrer Sound-Programme stehen. Sie erleichtern die Handhabung.

Das nächste Programmbeispiel soll Ihnen die Funktion des

Hüllkurvengenerators verdeutlichen. Übernehmen Sie der Einfachheit halber die Zeilen 10 bis 80 aus dem vorigen Programm. In dem Folgenden sind sie auch weggelassen.

```
100 A=9: D=9: S=8: R=9: H=400
110 POKE S1+5,16*A+D: POKE S1+6,16*S+R
120 POKE RS,0: POKE PL,15
130 POKE S1,37: POKE S1+1,17
140 POKE S1+4,33
150 FOR I=0 TO H: NEXT
160 POKE S1+4,32
```

Sie sollten nun die Werte in Zeile 100 verändern, damit Sie ein Gefühl dafür bekommen, welchen Einfluß die einzelnen Parameter auf den Ton ausüben.

Die Werte für A,D,S und R dürfen sich nur im Bereich 0 bis 15 bewegen, sonst gibt es einen illegal quantity error.

Wie Sie sicher herausgefunden haben, bestimmt A die Anstiegszeit, D die Abfallzeit auf den Haltepegel, H die Haltezeit des Tones auf dem Pegel S und R die Abfallzeit auf null nach rücksetzen des KEY-Bits.

Wenn der Wert für R mindestens 1 ist, sollten sie den Ton nicht, wie in den Programmbeispielen im CBM-Handbuch auf den Seiten 80 bis 90 dargestellt, durch den Wert 0 im Reg.4 abschalten. Dabei bricht der Ton nämlich sofort ab und der Parameter R hat keinerlei Einfluß.

Sie sollten vielmehr das Bit 0 gezielt rücksetzen, indem Sie das Register noch einmal mit dem ursprünglichen Wert laden, jedoch ohne Bit 0 (der Wert muß deshalb eine gerade Zahl sein).

Nun ein Dreiklangakkord auf dem Spinett. Behalten Sie wieder die Zeilen 10 bis 80 aus dem ersten Programm'

```
100 A=0: D=1: S=13: R=10: H=100
110 POKE S1+5,16*A+D: POKE S1+6,16*S+R
120 POKE S2+5,16*A+D: POKE S2+6,16*S+R
130 POKE S3+5,16*A+D: POKE S3+6,16*S+R
140 POKE RS,0: POKE PL,15
150 POKE S1,37: POKE S1+1,17
160 POKE S2,154: POKE S2+1,21
170 POKE S3,177: POKE S3+1,25
180 POKE S1+4,33: POKE S2+4,33: POKE S3+4,33
190 FOR I=0 TO H: NEXT
200 POKE S1+4,32: POKE S2+4,32: POKE S3+4,32
```

Im nächsten Beispiel wird die Frequenz des Tones in Abhängigkeit vom Verlauf der Hüllkurve verändert. Hierzu wird die Stimme 3 benutzt, da nur hier die Hüllkurve ausgelesen werden kann.

Experimentieren Sie auch hier mit den Werten in Zeile 100.

```
100 A=9: D=9: S=9: R=9: H=30
110 POKE RS,0: POKE PL,15
120 POKE S3+5,16*A+D: POKE S3+6,16*S+R
130 POKE S3+4,33
140 FOR I=0 TO H: POKE S3+1,PEEK(54300): NEXT
150 POKE S3+4,32
160 FOR I=0 TO R*4: POKE S3+1,PEEK(54300): NEXT
```



Lassen Sie zum guten Schluß noch das Raumschiff Enterprise  
akustisch Revue passieren:

```
100 A=15: D=0: S=8: R=13: H=8000
110 POKE RS,0:POKE PL,15
120 POKE S1,0: POKE S1+1,30
130 POKE S2,0: POKE S2+1,1
140 POKE S3,0: POKE S3+1,100
150 POKE S1+5,16*A+D: POKE S1+6,16*S+R
160 POKE S1+4,129: POKE S3+4,23
170 FOR I=0 TO H: NEXT
180 POKE S1+4,128: POKE S3+4,16
```

Sicherlich bieten Ihnen die vorstehenden Beispiele genügend  
Anregungen für eigene Versuche mit dem Synthesizer.  
Viel Spaß dabei.

## 2.3 SYNTHY-64

### DER MUSIK & TON SYNTHESIZER FÜR DEN COMMODORE 64

Als man das erste Mal von dem neuen Commodore 64 und seinen Fähigkeiten von Ton und Farbe hörte, war man sicherlich erst einmal unheimlich begeistert. Dann, nach der Lektüre des Commodore Benutzer Handbuches, fragte man sich sicherlich, ob da nicht vielleicht ein paar Seiten in diesem Handbuch fehlten. Denn nichts war über die Programmierung von Farbe, Ton und Graphik gesagt worden. Und das, wo diese Attribute doch eigentlich die Krönung dieses Computers sind.

Sicherlich muß man in diesem Zusammenhang auch sagen, daß die Programmierung von Farbe, Ton und Graphik, nicht unbedingt als leicht zu bezeichnen sind - denn man muß z.B. bei der Tonerzeugung, jedes einzelne Register für nur einen Ton unterschiedlich ansprechen. So benötigt man alleine für den Ton "C" drei POKE-Befehle:

POKE 54272,37: POKE 54273,17: POKE 54276,65

Dies ist für eine einfache Programmierung eines längeren Liedes natürlich ungeeignet. In Schleifen mußten Sie die Erzeugung von 4tel, 8tel usw. Noten steuern. Das alles konnten einem die Freude an seinem Commodore 64 schon etwas trüben. Zum Glück: gibt es Hilfsmittel, die Ihnen die ganze Arbeit mit PEEKs und POKEs und der aufwendigen Maschinenprogrammierung ersparen.

Ein solches Hilfsmittel ist zum Beispiel das Programm SYNTHY 64. Es erlaubt die Ausnutzung der fantastischen Soundmöglichkeiten des Commodore 64 auch unter einer einfachen BASIC Programmierung. Am Beispiel von SYNTHY 64 - Sie erhalten dieses von DATA BECKER vertriebene Programm bei Ihrem örtlichen Commodore Fachhändler - wollen wir zeigen, wie leicht und komfortabel die Musikprogrammierung des Commodore 64 sein kann.

Dieses Programm erlaubt das Spielen von mehreren Stimmen, die Anzeige des momentan gespielten Tones, das Abspeichern und Laden der verschiedenen Lieder und vieles mehr.

Wir wollen in diesem Buch auch auf dieses Programm eingehen, da es doch die Programmierung von Musik sehr vereinfacht.

Die Erzeugung der Töne geschieht hier wie man es von dem normalen Notenblatt gewohnt ist - und das in Zusammenhang mit einer BASIC ähnlichen Struktur. Sie geben also eine Zeilennummer vor und danach die Noten oder Befehle ein.

Noten können genau nach ihrem Namen eingegeben werden. Also wird zum Beispiel die Note "C" ganz einfach als "C" eingegeben.

Bei der ersten Note jeder Stimme müssen Sie folgendes mit angeben:

STIMME - NOTE - OKTAVE - / - DAUER

also zum Beispiel: +G5/4. + steht hier für das Instrument Akkordeon, dann die Note "G", gespielt in der fünften Oktave im 1/4 Takt. Solange Sie diese Werte nicht ändern, brauchen Sie nur die Note anzugeben.

Das Grundprogramm, das immer zu dem eigentlichen Musikeil gehört, sieht folgendermaßen aus:

```
1 RUN"<CLR>"          <CLR> erscheint als reverses Herz
63000 REM
63005 REM*****
63010 $WP $PB $AO $D9 $S0 $R0 $F0 RETURN PIANO
63020 $WT $A4 $D2 $S10 $R5 $F0 RETURN FLOETE
63030 $WS $A6 $D0 $S10 $R1 ZB X10 Y12 $F1 RETURN TROMPETE
63040 $WP $P1 $AO $D9 $S0 $R0 $F0 RETURN BANJO
63050 $WS $A6 $D5 $S2 $R2 ZH X8 Y12 $F1 RETURN AKKORDEON
```

(ACHTUNG: Unser Drucker beherrscht den Klammeraffen in Verbindung mit deutschen Zeichen leider nicht - Wir verwenden hierfür das Paragrafenzeichen)

Dieses Programm brauchen Sie nicht einzugeben. Sie starten ganz einfach den SYNTHY 64, laden das Demoprogramm, unterbrechen dieses Programm und drücken die SHIFT und C-Taste gleichzeitig. Dies löscht den Musikspeicher, behält aber das Grundprogramm. Nun können Sie ihre Komposition eingeben.

Der RUN-Befehl in Zeile 1 MUSS der erste Befehl in jedem Musikprogramm sein, das Sie schreiben wollen. Die Zeilen 63010 bis 63050 sind fünf, voneinander unabhängige Tonerzeugungsroutinen, die von Ihnen aufgerufen werden können. Sie brauchen dafür dem Programm am Anfang nur mitzuteilen, daß es, wenn es an eine bestimmte Stimme kommt (erinnern Sie sich an den Aufbau eines Tones), zu der entsprechenden Unterroutine zu gehen. In unserem Beispiel, wären wir also in die Unterroutine in Zeile 63050 (Akkordeon) verzweigt.

Die Befehle, die Sie in den Zeilen 63010 bis 63050 sehen, sind Besonderheiten des SYNTHY 64. Durch diese Befehle werden die entsprechenden Register im Commodore 64 geladen.

ACHTUNG: Diese Befehle laufen nur im Zusammenhang mit dem SYNTHY 64, und führen ohne dessen Verwendung zu einem Fehler.

Es handelt sich bei diesen Befehlen, um die unterschiedliche Ansteuerung von Generator, Filter, Ringmodulator und um die unterschiedlichen Frequenzen. So können unterschiedliche Wellenformen gesetzt werden, die Frequenz kann frei gewählt und die Filter unterschiedlich geschaltet werden.

Auf der nächsten Seite zeigen wir Ihnen an einem bekannten Lied ("Swing low - sweet chariot"), wie die Programmierung eines kompletten Liedes aussehen kann.

# SWING LOW - SWEET CHARIOT

```

1 RUN"<CLR>"
10 +GOSUB 63050 GOSUB63020
50 T130
100 +B5/4 ♯G4/1 +G4/2 B5/4 G4/4. ♯+G +G/8 E/8 +D/4.
150 G/8 ♯+G +G G B5 B B/4 ♯+D +D/1
200 E/8 ♯+G +D B/2 D/4 G4/4. ♯+C/2 +G/8 E ♯+G +D/4
250 G/8 ♯+G +G G B5 ♯+D +B A/4 G4/2. ♯+G/1 +B5/4
300 D ♯+G +G4/8 E G G/4 G/8 G ♯+C/2 +G G/4 E/8 ♯+G +D/4.
350 G/8 ♯+G/1 +G G B5 B D/4 D/2. ♯+D +D/4
400 E/8 ♯+G +D B/4 B G4 G/8 ♯+C/2 +G G G E ♯+G/2 +D/4.
450 A/8 ♯+G +G G B5 ♯+D +B A/4 G4/1 ♯+G
999 END
63000 REM
63005 REM*****
63010 $WP $PB $AO $D9 $S0 $R0 $FO RETURN PIANO
63020 $WT $A4 $D2 $S10 $R5 $FO RETURN FLOETE
63030 $WS $A6 $D0 $S10 $R1 ZB X10 Y12 $F1 RETURN TROMPETE
63040 $WP $P1 $AO $D9 $S0 $R0 $FO RETURN BANJO
63050 $WS $A6 $D5 $S2 $R2 ZH XB Y12 $F1 RETURN AKKORDEON

```

Wenn Sie dieses Programm eingetippt haben, geben Sie ganz einfach den RUN-Befehl ein, und schon hören Sie Ihr erstes Lied. Viel Vergnügen beim Komponieren.

Selbstverständlich können Sie auch ihre eigenen Hilfsprogramme zur Erzeugung von Tönen auf dem Commodore 64 erstellen. Dafür sollten Sie sich zunächst einmal ausführlich mit dem ganzen Kapitel 4 auseinandersetzen.

Wichtig ist vor allen Dingen, daß Sie die Benutzung der Register des Sound-Controllers verstanden haben.

## Kapitel 3 DIE GRAPHIK UND IHRE PROGRAMMIERUNG

### 3.1 Der Video-Controller 6569 VIC

#### 3.1.1 Allgemeines über den VIC 6569

Der VIC ist ein sehr ausgeklügeltes neues IC zur Bereicherung der 65xx-Familie.

Er erfüllt nicht nur alle zur Erzeugung eines Bildes notwendigen Aufgaben, sondern führt auch zur Entlastung des Prozessors das gesamte Timing für einen dynamischen Speicher aus.

Das Wichtigste in Kürze:

- \* 16 Farben
- \* 40 Zeichen / 25 Zeilen
- \* Hi-Res-Grafik mit 320 x 200 Punkten
- \* 5 Betriebsarten
- \* Handhabung von 8 Sprites mit je 24 x 21 Punkten
- \* Erzeugung eines normgerechten PAL-Signals
- \* Selbständiges Handling von 16K dyn. Ram
- \* Verschiebbarer Zeichengenerator
- \* Verschiebbarer Video-Ram

#### Pinbeschreibung des VIC 6569

- |       |  |
|-------|--|
| 1 -7  | D6-D0; Prozessordatenbus   |
| 8     | -IRQ; 0 wenn ein Bit des IMR und des IRR übereinstimmen          |
| 9     | -LP; Eingang, Light-Pen-Strobe                                   |
| 10    | -CS; Prozessorbustaktionen finden nur bei CS=0 statt.            |
| 11    | R/-W; 0=Übernahme der Daten vom Bus.                             |
| 12    | BA; =0 wenn Daten bei einem Lesezugriff noch nicht bereitstehen. |
| 13    | VDD; +12V  |
| 14    | COLOR; Farbinformation Ausgang                                   |
| 15    | SYNC; Zeilen- und Bildsynchronisationsimpulse                    |
| 16    | AEC; 0=VIC benutzt Systembus, 1=Bus frei                         |
| 17    | 0OUT; Ausgang Systemtakt   |
| 18    | -RAS; dyn. Ram Steuersignal                                      |
| 19    | -CAS; wie oben   |
| 20    | GND  |
| 21    | 0COLOR; Eingang Farbfrequenz                                     |
| 22    | 0IN; Eingang Dotfrequenz   |
| 23    | A11; Prozessoradressbus  |
| 24-29 | A0/A8-A5/A13; gemultiplexer (Video-) Ram-Adressbus               |
| 30-31 | A6-A7; (Video-) Ram-Adressbus                                    |
| 32-34 | A8-A10; Prozessoradressbus                                       |
| 35-38 | D11-B; Daten aus Farbram   |
| 39    | D7; Prozessordatenbus  |
| 40    | VCC; +5V   |

### 3.1.2 Registerbeschreibung des VIC

Der VIC verfügt über 47 Register, die im Folgenden beschrieben werden.

#### REG 0 Sprite-Register 0 X-Koordinate

Hier sind 8 Bits der Bildschirmkoordinate X enthalten, auf der das Sprite dargestellt wird. Das 9. Bit befindet sich in REG 16.

#### REG 1 Sprite-Register 0 Y-Koordinate

Wie oben, jedoch für die Y-Richtung. Dieses Register hat keinen Übertrag.

Die Register 2 bis 15 folgen alle dem oben beschriebenen Aufbau.

Registerpaar 2/3 ist für Sprite 1, Paar 4/5 für Sprite 2 usw.

Es geht weiter mit

#### REG 16 MSB der X-Koordinaten

Hier befinden sich die Überläufe aus den Sprite-X-Registern, und zwar Bit 0 für Sprite 0, Bit 1 für Sprite 1 usw.

#### REG 17 Steuerregister 1

Bit 0-2 Offset der Darstellung vom oberen Bildrand in Rasterzeilen

Bit 3 0=24 Zeilen, 1=25 Zeilen

Bit 4 0=Bildschirm aus

Bit 5 1=Standard Bitmap Mode

Bit 6 1=Extended Colour Mode

Bit 7 Übertrag aus REG 18

REG 18 Hier wird die Nummer der Rasterzeile angegeben, bei deren Strahldurchlauf ein IRQ ausgelöst werden kann. Übertrag dieses Registers in REG 17

REG 19 X-Anteil der Bildschirmposition, an der sich der Strahl gerade befand, als ein Strobe ausgelöst wurde (Pin LP=0).

REG 20 Wie oben, jedoch Y-Anteil.

#### REG 21 Sprite Enable

Jedem Sprite ist ein Bit zugeordnet. 1=Sprite an, 0=Sprite aus.

#### REG 22 Steuerregister 2

Bit 0-2 Offset der Darstellung vom linken Bildrand in Rasterpunkten

Bit 3 0=38 Zeichen, 1=40 Zeichen

Bit 4 1=Multi Color Mode

#### REG 23 Sprite Expand X

Jedem Sprite ist ein Bit zugeordnet. 1=Sprite wird doppelt so breit.

- REG 24 Basisadresse von Zeichengenerator und Videoram  
 Bit 1-3 Adressbits 11-13 für die Zeichenbasis  
 Bit 4-7 Adressbits 10-13 für die Basis des Videoram  
 Die im CBM 64-Handbuch auf Seite 158 angegebene Belegung dieses Registers ist falsch!
- REG 25 IRR Interrupt Request Register  
 Bit 0 Auslöser ist REG 10  
 Bit 1 Auslöser ist REG 31  
 Bit 2 Auslöser ist REG 30  
 Bit 3 Auslöser ist Pin LP  
 Bit 7 =1 wenn mind. eines der anderen Bits =1 ist.
- REG 26 IMR Interrupt Mask Register  
 Belegung wie oben. Bei Übereinstimmung mind. eines Bits aus IRR und IMR wird der Pin IRQ=0.
- REG 27 Jedem Sprite ist ein Bit zugeordnet. 1=Hintergrundzeichen hat vor dem Sprite Priorität.
- REG 28 Jedem Sprite ist ein Bit zugeordnet. 1=Sprite Multi Color Mode.
- REG 29 Sprite Expand Y  
 Jedem Sprite ist ein Bit zugeordnet. 1=Sprite wird doppelt so hoch.
- REG 30 Sprite-Sprite Collision  
 Jedem Sprite ist ein Bit zugeordnet. Berührt ein Sprite ein anderes, so werden die entsprechenden Bits =1. Gleichzeitig wird IRR Bit 2 =1. Nach dem Ereignis muß dieses Register gelöscht werden, da sich die Bits nicht selbsttätig zurücksetzen.
- REG 31 Sprite-Background Collision  
 Wie oben, jedoch tritt das Ereignis ein, wenn ein Sprite Berührung mit einem Hintergrundzeichen hat.
- REG 32 Exterior Color (Rahmenfarbe)
- REG 33-36 Background Color 0-3 (Hintergrundfarben)
- REG 37-38 Sprite Multi Color 0-1
- REG 39-46 Color Sprite 0 - Sprite 7

### 3.1.3 Die Betriebsarten des VIC

Der VIC kennt drei Grundarten der Darstellung: Zeichen aus einem Generator, Einzelpunktdarstellung und die Darstellung von Sprites.

#### Sprites:

Sprites können in ihrer Position verändert werden. Dazu dient ein Registerpaar. Die Auflösung der Sprite-Bewegung beträgt 312 x 256 Punkte. Durch diesen großen Bereich können

Sprites auch aus dem Bildschirm hinausbewegt werden.  
Die Farbwiedergabe der Sprites ist auf zwei Arten möglich:  
Normale Auflösung mit zwei Farben, und halbe Auflösung mit vier Farben.

Im ersten Fall hat das Sprite eine Auflösung von 24x21 Punkten. Die eine Farbe stammt aus dem Sprite Color Register, die andere Farbe ist transparent, d.h. sie ist gleich der Hintergrundfarbe.

Im zweiten Fall (Multi Color Mode: das dem Sprite entsprechende Bit in REG28 muß gesetzt sein) hat das Sprite nur eine Auflösung von 12x21 Punkten. Hierbei bilden je zwei Bits des Sprite-Bitmusters einen Punkt auf dem Bildschirm. Diese beiden Bits bestimmen die Farbe des Punktes folgendermaßen:

00=transparent

01=Multi Color Register 0

11=Multi Color Register 1

10=Sprite Color

Damit der VIC weiß, woher er das Bitmuster nehmen soll, existiert für jedes Sprite ein Sprite-Pointer.

Diese acht Pointer belegen die obersten acht Bytes im Videoram.

Multipliziert man den Wert eines Pointerbytes mit 64, so erhält man die reale Speicheradresse, wo das Bitmuster abgelegt sein muß.

Normale Zeichendarstellung:

Bei dieser Betriebsart holt sich der VIC ein Byte aus dem Videoram.

Dieses benutzt er als Zeiger auf eine Position im Zeichengenerator. Das dort abgelegte Bitmuster erscheint schließlich auf dem Bildschirm.

Auf diese Weise können 256 verschiedene Zeichen dargestellt werden.

Die Farbinformation für das Zeichen befindet sich in einem besonderen Farbram. Für jede Bildschirmposition gibt es vier Bit im Farbram, die eine aus 16 Farben auswählen. Diese Farbe gilt aber nur für die gesetzten Punkte im Zeichenmuster. Die nicht gesetzten beziehen ihre Farbe aus dem Hintergrund-Farregister 0. Daher haben auch alle Zeichen die gleiche Hintergrundfarbe.

Normale Zeichendarstellung im Multi Color Mode:

(REG 22 Bit 4=1)

Bei dieser Betriebsart werden zunächst die vier Bits des Farbrams geprüft. Ist das höchstwertige Bit=0, so wird das Zeichen in der normalen 8x8-Matrix dargestellt. Die Farbe für die 1-Bits des Musters wird mit den übrigen drei Bits des Farbrams bestimmt. Die Farbe für die 0-Bits kommt wieder aus dem Background-Farregister.

Ist das höchstwertige Bit des Farbram jedoch =1, so stellen je zwei Bits des Zeichenmusters einen Punkt dar. Das Zeichen wird jetzt als 4x8-Matrix dargestellt mit doppelt breiten Punkten. Die Bitpärchen des Musters bestimmen jetzt die Farbe eines Punktes:

00=Hintergrund-Farregister 0

01=Hintergrund-Farregister 1

10=Hintergrund-Farregister 2

11=übrige drei Bits aus dem Farbram



In diesem Fall kann ein Zeichen vier Farben gleichzeitig haben.

#### Extended Color Mode:

(REG 17 Bit 6=1)

Diese Betriebsart ähnelt dem Normalmodus, da auch hier jedes Zeichen nur aus zwei Farben bestehen kann, jedoch ist die Hintergrundfarbe nicht notwendigerweise für jedes Zeichen gleich.

Für die 1-Bits des Musters stammt die Farbe nach wie vor aus dem Farbram. Für die 0-Bits aber wird die Farbe aus den beiden höchstwertigen Bits des Videoram abgeleitet:

00=Hintergrund-Farbregister 0

01=Hintergrund-Farbregister 1

10=Hintergrund-Farbregister 2

11=Hintergrund-Farbregister 3

Da jetzt aber nur noch sechs Bits als Zeiger auf das Zeichenumuster übrig bleiben, bleibt nur noch ein Vorrat von 64 Zeichen.

#### Normaler Einzelpunktmodus:

(REG 17 Bit 5=1)

In dieser Betriebsart besteht ein direkter Zusammenhang zwischen Speicher und Bildschirm. Ein Bit im Speicher entspricht einem Punkt auf dem Schirm.

Hierzu benötigt man einen Speicher von 8K. Der normale Videoram wird jetzt als Farbram mißbraucht. Der normale Farbram hat keine Funktion.

Die Auflösung beträgt 320x200 Punkte. Je ein Byte des Videorams liefert die Farbinformation für eine Gruppe von 8x8 Punkten, und zwar derart, daß das höherwertige Halbbyte für die Farbe der 1-Bits zuständig ist und das niederwertige Halbbyte für die Farbe der 0-Bits.

#### Mehrfarben-Einzelpunkt-Modus:

(REG 17 Bit 5=1 und REG 22 Bit 4=1)

Hier beträgt die Auflösung nur 160x200 Punkte, weil je zwei Bits des Speichers einem doppelt breiten Punkt auf dem Bildschirm zugeordnet sind.

00=Hintergrund-Farbregister 0

01=Videoram höherwertiges Halbbyte

10=Videoram niederwertiges Halbbyte

11=Farbram

Die Verschiebemöglichkeiten von Videoram und Zeichengenerator werden in den Abschnitten 3.3 und 3.4 behandelt.

### 3.2 Die Schnittstelle zum Prozessor

Der VIC ist wie ein normales Peripherieelement am Prozessor angeschlossen. Er kann ohne Einschränkung gelesen und beschrieben werden. Dennoch hat er das gesamte Systembus-Timing in der Hand.

Er ermöglicht Zugriffe oder verhindert sie, bedient den dynamischen Ram, usw.

Dies ist möglich, weil der Systemtakt im VIC erzeugt wird.

Er weiß deshalb jederzeit, wann welche Transaktionen stattfinden können. Auch kann er die Steuerung des Bus durch den Prozessor unterbinden. Das ist immer dann nötig, wenn VIC auf Videoram, Charactergenerator oder Farbram zugreifen muß. Diese zyklischen Zugriffe müssen sein, damit kontinuierlich ein Bild auf dem Schirm erscheint.

Um den Prozessor nicht bei seiner Arbeit zu stören, nutzt er geschickt die Taktlücken aus, in denen der Prozessor den Bus nicht benötigt.

Die Vereinbarung ist einfach folgende, daß immer wenn  $O2=1$  ist, der Prozessor den Bus belegt, und bei  $O2=0$  der VIC. Um den Bus auch physikalisch freizumachen, legt VIC den Pin AEC auf 0, worauf der Prozessor den Bus seinerseits in den hochohmigen Zustand (Tri-State) bringt.

Auf diese Weise stört keiner den anderen.

### 3.3 Die Schnittstelle zum Ram

Zur Verdeutlichung des hier Gesagten dient das Blockschema auf der nächsten Seite.

Der VIC kann von Haus aus nur 16K Ram adressieren.

Deshalb muß er, und das ist wichtig, beim Verschieben von Videoram oder Zeichengenerator, die beiden fehlenden Adressbits von außerhalb zu Hilfe nehmen.

Diese Bits werden von PA0-1 der CIA 2 zur Verfügung gestellt. Verändert man diese Bits, so verschiebt sich der Videoram gleich in 16K-Schritten. Diese Bits stellen also die Adressbits 14 und 15 des Videorams dar.

Unter der Voraussetzung, daß man über genügend freien Speicher verfügt, kann man mit diesen Bits mehrere Bildschirmseiten umschalten.

Beachten Sie hierbei bitte, daß diese Bits L-aktiv sind, d.h. um die unterste 16K-Seite zu selektieren, müssen beide Bits in der CIA 2 auf 1 gesetzt sein. Für die oberste Seite müssen beide 0 sein.

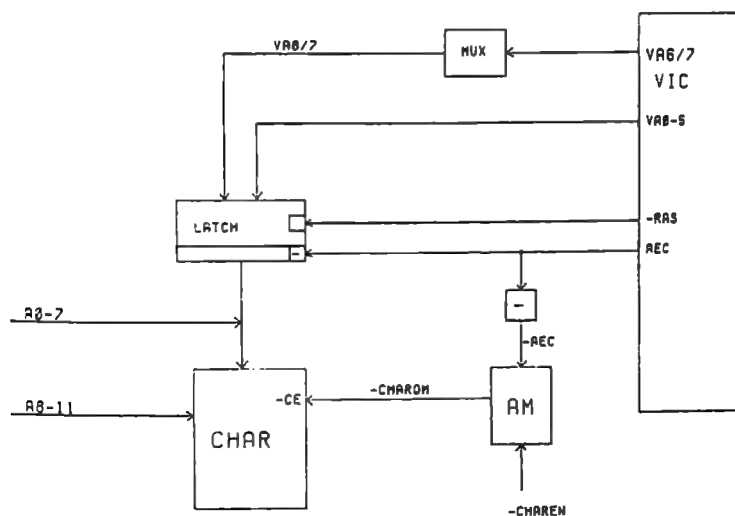
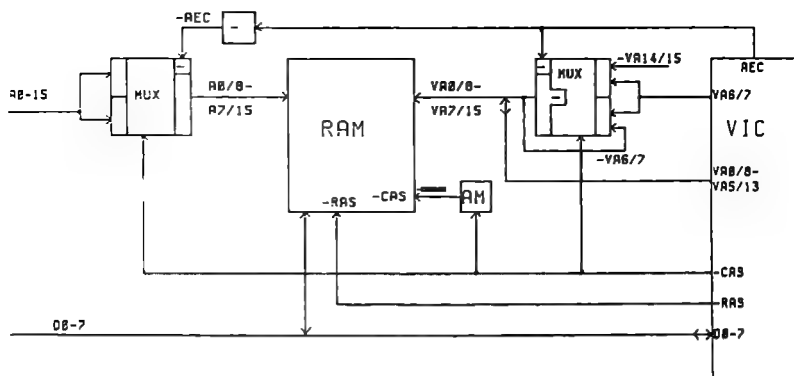
Im Hi-Res-Modus könnte zum Beispiel die eine Bildschirmseite dargestellt werden, während eine andere unsichtbar bereits aufbereitet wird.

In feineren Schritten von 1K kann man den Videoram mit Hilfe des VIC-Registers 24 verschieben. Hier bedeuten die Bits 4-7 die Ramadressen 10-13.

### 3.4 Die Charactergenerator-Schnittstelle

Das Bild auf der übernächsten Seite illustriert die folgende Erläuterung.

Ähnlich wie im vorigen Abschnitt ist der Ablauf auch hier. Die CIA-Bits verschieben gleichzeitig mit dem Videoram auch den Charactergenerator in ebensolchen Schritten.



Mit Hilfe der Bits 1-3 im REG 24 des VIC kann er in 2K-Schritten verschoben werden. Sie stellen die Adressbits 11-13 dar.

Der im CBM 64 eingebaute Character-Rom besitzt dank des Adressmanagers einen Sonderstatus. Dieses wird vom VIC nur angesprochen, wenn sich seine relative Adresse zwischen \$1000-\$1FFF, bzw. \$9000-\$9FFF bewegt.

Tatsächlich liegt der Generator im Prozessoradressraum bei \$D000.

Da der Character-Rom rein physikalisch nicht im Zugriffsbereich aller Videoadressen liegt, müssen die fehlenden Bits in einem besonderen Puffer zwischengespeichert werden. Es sind dies die Adressbits 0-7. Die Charactergeneratorbasis im REG 24 spielt noch eine weitere Rolle. Sie legt nämlich nicht nur die Startadresse des Generators fest, sondern Bit 3 bestimmt im Einzelbit-Modus auch die Lage des Bildschirmspeichers. Hierbei bleiben die Bits 1 und 2 unberücksichtigt.

### 3.5 Die Schnittstelle zum Farb-Ram

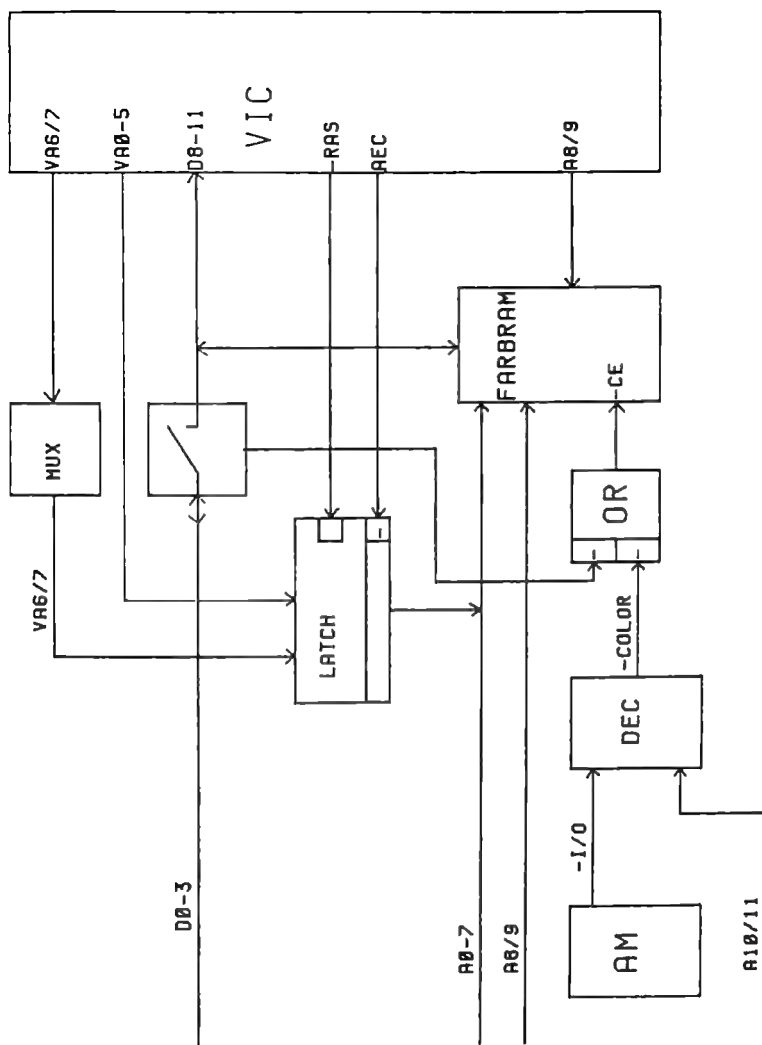
Das Farbram ist, im Gegensatz zu Videoram und Zeichengenerator, nicht verschieblich. Die Adressierung kann, ebenso wie beim Character-Rom, von zwei Seiten erfolgen, nämlich vom Prozessorbus und vom VIC-Bus. Auch hier fehlen dem VIC zur Adressierung wieder die Bits 0-7. Sie werden auf die gleiche Weise gewonnen wie beim Character-Rom.

Zum VIC hin hat der Farbram eine eigene Datenschnittstelle. Sie ist mittels eines Analogschalters vom Datenbus des Prozessors trennbar. Die Angelegenheit funktioniert denkbar einfach:

Bei AEC=0 (also VIC steuert den Bus) liegt der Schalter in Richtung VIC. Bei AEC=1 in der anderen Richtung.

Darüber hinaus wird der Schreib/Lesezugriff seitens des Prozessors auf den Farbram vom Adress-Manager gesteuert.

Das Blockscheema auf der nächsten Seite soll diese Umstände noch anschaulicher machen.



### 3.6 Programmierung von Farbe und Graphik

Die Stärken des Commodore 64, sind sicherlich seine Farbe und sein eingebauter Synthesizer. Aber wie soll man seine Stärken nutzen, wenn man nirgends erfährt, wie Graphik, Sprites und Ton zu programmieren sind ?

Auf den nächsten Seiten, wollen wir Ihnen nun erklären, wie man diese Commodore 64 Besonderheiten richtig anwendet, und somit auch die ganze Stärke dieses Rechners ausnutzen kann.

Die Graphikmöglichkeiten beim Commodore 64 sind stärker, als bei den meisten anderen vergleichbaren Computern dieser Frei- und Leistungsklasse. Auch hat er eine wesentlich bessere Graphik als sein kleiner Bruder, der VC-20.

Die wichtigste Besonderheit des Commodore 64, ist die hardwaremäßige Möglichkeit, bis zu 8, frei definierbaren Sprites, auf dem Bildschirm zu bewegen. Jedes dieser sogenannten SPRITES kann aus einer anderen Farbe bestehen. Es stehen insgesamt 16 Farben zur Verfügung. Zusätzlich sind MULTICOLOR Sprites möglich. Diese Sprites können aus 3 Farben bestehen. Ein Sprite besteht aus einer 24 x 21 Punkte Matrix, vergleichbar mit einem Block aus 3 x 3 Buchstaben.

Man kann jederzeit kontrollieren, ob Sprites auf dem Bildschirm zusammengestoßen sind, und wo sie sich befinden. Aber zu der Programmierung der Sprites kommen wir später noch. Nun wollen wir uns erst einmal mit der Programmierung von Farbe und Graphik befassen.

#### GRAPHIK UND FARBE

Sie können beim Commodore 64 zwischen 12 Farben wählen. Diese Farben können im Hintergrund und Vordergrund beliebig verändert werden. Die Änderung der Schriftfarbe kann auch von der Tastatur bewerkstelligt werden. Durch gleichzeitiges Drücken der CTRL-Taste oder der C-Taste (dem CBM Symbol, auf der Tastatur links unten) und einer Taste zwischen 1 und 8, werden folgende Schriftfarben erzeugt:

Taste -----	Farbe -----	Kennzahl -----
CTRL - 1	SCHWARZ	0
CTRL - 2	WEISS	1
CTRL - 3	ROT	2
CTRL - 4	TÜRKIS	3
CTRL - 5	VIOLETT	4
CTRL - 6	GRÜN	5
CTRL - 7	BLAU	6
CTRL - 8	GELB	7
C= - 1	ORANGE	8
C= - 2	BRAUN	9
C= - 3	HELL ROT	10
C= - 4	GRAU 1	11
C= - 5	GRAU 2	12
C= - 6	HELL GRÜN	13
C= - 7	HELL BLAU	14
C= - 8	GRAU 3	15

Es kann vorkommen, daß die Schrift auf Ihrem Bildschirm nur schwer, oder überhaupt nicht zu lesen ist. Das liegt an dem Kontrast zwischen der Schriftfarbe zum Vorder- und Hintergrund. So sieht beispielsweise eine grüne Schrift auf schwarzem Vorder- und Hintergrund sehr gut aus. Aber experimentieren Sie ruhig ein wenig, Sie werden dann selbst "Ihre" Schrift finden.

Der Vorder- und Hintergrund lassen sich leicht durch Eingabe der entsprechenden Farbwerte an die Speicherstellen 53280 und 53281 verändern. In diesen beiden Adressen stehen immer die momentan benutzten Bildschirmfarben. So bewirkt zum Beispiel:

```
POKE 53280,0
POKE 53281,0
```

einen völlig schwarzen Bildschirm. Die Schrift bleibt aber weiterhin in ihrer normalen Farbe. Um die Farbe von nur einem Bildpunkt zu ändern, genügt ein:

```
POKE 55296,0
```

Dieser POKE-Befehl ändert die Farbe eines Zeichens in der linken, oberen Bildschirmecke in schwarz. Alle anderen Zeichen behalten ihre normale Farbe. Diese Adresse ist die erste Adresse im Farb-RAM des Commodore 64. In diesem besonderen RAM befinden sich die Farben aller Zeichen, die auf dem Bildschirm dargestellt sind. Der ganze Farb-RAM geht von 55296 bis 56295. Für jede der 1000, auf dem Bildschirm darstellbaren Zeichen, ist also ein Byte reserviert. Die Adresse 55296 bezieht sich dabei auf Zeile 1, Spalte 1, die Adresse 55297 auf Zeile 1, Spalte 2 usw. Die Adresse 56295 ist dann die Information von Zeile 25, Spalte 40.

Es gibt aber zur Zeichendarstellung noch einen anderen Speicher: Der Bildschirmspeicher. In diesem Speicher befindet sich das tatsächliche Zeichen, ohne Angabe der Farbe. Der RAM-Bereich liegt bei 1024 bis 2023.

Um auf dem Bildschirm lauter schwarzer 'A's darzustellen, könnte man folgendes kleine Programm verwenden:

```
10 PRINT CHR$(147);: REM LÖSCHEN DES BILDSCHIRMS
20 FOR I=1024 TO 2023: REM BILDSCHIRMSPEICHER
30 POKE 54272+I,0: REM FARBSPEICHER (54272+1024=55296)
40 POKE I,1: REM 1 = KENNZAHLE FÜR A, 2 = B USW.
50 NEXT: REM FÜLLE GANZEN BILDSCHIRM- UND FARBRAM
60 GOTO 60: REM DAMIT DER BILDSCHIRM NICHT ÜBERSCHRIEBEN WIRD
```

Sie sehen dann, wie sich der Bildschirm von links oben nach rechts unten mit lauter 'A's füllt.

Um noch Farbe in die ganze Sache zu bringen, könnte man die Zeile 30 abändern:

```
30 POKE 54272+I,INT(RND(1)*16)
```

Nun läuft das Programm genau so wie die erste Version ab, mit dem Unterschied, daß jedes Zeichen in einer zufällig gewählten Farbe ausgegeben wird. An dieser Stelle noch etwas über die ZUFALLSZAHLN. Nicht jeder ist damit vertraut, obwohl man diese Funktion doch recht oft benötigt. Besonders bei der Programmierung von Spielen, kann man eigentlich überhaupt nicht

auf die RND Funktion verzichten. Die Benutzung ist denkbar einfach. Wollen Sie zum Beispiel eine Zufallszahl zwischen 0 und 15 generieren, um zum Beispiel eine Zufallsfarbe zu erzeugen, so genügt die oben angeführte Sequenz (INT(RND(1)\*16)).

Aber damit sind natürlich die Graphikmöglichkeiten des Commodore 64 noch lange nicht erschöpft. Die Benutzung der hochauflösenden Farbgaphik, läßt sich am besten am Beispiel eines PLOT Programms zeigen. Dieses Programm finden Sie am Ende dieses Kapitels. Es zeigt Ihnen die Verwendung einiger interessanter Graphik-adressen und ihre Wirkung.

Aber zunächst, wollen wir erst einmal ein paar Fakten über die Graphikprogrammierung anführen.

Die Programmierung von Kreisen oder Linien zwischen 2 verschiedenen Punkten, ist beim Commodore 64 leider nicht so komfortabel, wie bei einigen anderen Computern. So gibt es zum Beispiel keinen Befehl, um eine Gerade von Pnkt X nach Punkt Y zu ziehen. Oder ein Befehl, um einen Kreis mit einem bestimmten Radius an einem bestimmten Ort zu zeichnen. Da diese Befehle nicht existieren, heißt natürlich nicht, daß der Commodore 64 dieses nicht kann. Man muß es ihm nur beibringen.

Am Ende des Kapitels über Graphik, finden Sie ein ausführliches Hilfsprogramm zur Erstellung von Graphiken. Wir nennen es HI-RES GRAPHIK-HILFE. Mit Hilfe dieses Programmes können Sie schnell und bequem einfache Graphiken erzeugen. Wollen Sie aber kompliziertere Graphiken darstellen, so haben wir noch ein anderes Programm entwickelt. Das Programm heißt SUPERGRAPHIK 64, und ist bei ihrem örtlichen Commodore Fachhandler zu bekommen. Aber zunächst können Sie sich ruhig an der Benutzung unserer GRAPHIK-HILFE üben. Sie werden sehen, wie schnell sich eine gute Graphik auf dem Commodore 64 erstellen läßt.

Nun aber zu dem oben erwähnten Plot Programm. Aufgabe dieses Programmes ist zum einen, die Erstellung einer Sinuskurve auf dem Bildschirm, und zum anderen, Ihnen die Anwendung der Commodore 64 Graphik zu demonstrieren.

Falls Sie sich über die Bedeutung der verschiedenen Register, die wir in diesem Programm verwenden, informieren wollen, so können Sie dies im Kapitel mit dem Registerplan des Graphikprozessors tun.

```
10 REM SINUS-PLOT PROGRAMM
20 V=53248: REM STARTADRESSE DES GRAPHIK-PROZESSORS
30 AD=8192: REM STARTADRESSE DER HI-RES BIT MAP
40 POKE V+17,59: REM EINSCHALTEN DER GRAPHIK
50 POKE V+24,24: REM "
60 FOR I=1024 TO 2023: REM SETZEN DES HI-RES FARBRAM
70 : POKE I,16: REM FARBKENNZAH
80 NEXT I
90 FOR I=8192 TO 16383: REM LÖSCHEN DER HI-RES BIT MAP
100 : POKE I,0
110 NEXT I
120 FOR X=0 TO 319: REM ZEICHNEN DER X-ACHSE
130 : Y=100: REM POSITION DER X-ACHSE
140 : GOSUB 1000: REM AUFRUF DER ZEICHEN-ROUTINE
150 NEXT X
160 FOR Y=0 TO 199: REM ZEICHNEN DER Y-ACHSE
```



```

170 : X=160: REM POSITION DER Y-ACHSE
180 : GOSUB 1000: REM AUFRUF DER ZEICHEN-ROUTINE
190 NEXT Y
200 X=0
210 FOR I=-3.14159265 TO 3.14159265 STEP 0.0196349541
220 : REM INTERVALLGRENZEN
230 : Y=100+99*SIN(I): REM FUNKTION
240 : GOSUB 1000
250 : X=X+1
260 NEXT I
270 GOTO 270: REM DAMIT DER BILDSCHIRM NICHT VERÄNDERT WIRD
1000 OY=320*INT(Y/8)+(Y AND 7): REM BERECHNEN DES PUNKTES
1010 OX=8*INT(X/8)
1020 MA=2^((7-X) AND 7)
1030 AV=AD+OY+OX
1040 POKE AV,PEEK(AV) OR MA: REM PLOTTEN DES PUNKTES
1050 RETURN

```

Wenn Sie dieses Programm vom Aufbau her verstanden haben, werden auch Sie bald Ihre eigenen mathematischen Funktionen auf diese Art und Weise darstellen können.

Schreiben Sie vielleicht doch selber mal einen Funktionsplotter - also ein Programm, das nach Eingabe der Funktion und deren Grenze die Zeichnung auf Bildschirm oder Drucker gibt. Zum Thema Drucker und Graphik wollen wir hier noch anführen, daß eine Übertragung der Graphik vom Bildschirm auf den Drucker, nicht besonders einfach ist. Sie finden in diesem Buch zwei Programme, mit denen Sie ein sogenanntes HARDCOPY erstellen können. Die erste Version, für eine Hardcopy auf EPSON Druckern, befindet sich im Programm HI-RES GRAPHIK-HILFE, die andere Version, für ein einfaches Text-Hardcopy, finden Sie im Kapitel 6.3.2 unter dem Namen HARDCOPY.

Was geschieht denn nun alles in unserem Plot-Programm? Zunächst werden in den Zeilen 20 und 30 die beiden Startadressen für die Graphikseite definiert. Dann wird mit den beiden POKE Befehlen

POKE V+17,59 und POKE V+24,24

die Graphik eingeschaltet, oder besser gesagt von Textmodus auf Graphikmodus umgeschaltet. Dies ist notwendig, da der Computer sonst davon ausgeht, daß Sie weiterhin nur im Textmodus, also dem normalen Modus, arbeiten wollen. Wie Sie wissen, werden sowohl Text als auch Graphik in dem jeweiligen Bildschirmspeicher abgespeichert. Der Commodore 64 kann natürlich nur eine der beiden Speicher auf einmal darstellen. Würden Sie also, von dieser Idee ausgehend, im Textmodus einen Graphik bearbeiten, so würde der Computer überhaupt nichts anzeigen. Umgekehrt, daß heißt also Text im Graphikmodus zu erzeugen, werden allerdings eigenartige Ergebnisse erzeugt. Darauf kommen wir gleich noch.

Wenn Sie später wieder auf die Textseite umschalten wollen, müssen Sie wieder die beiden Adressen (V+17, V+24) auf ihren alten Wert setzen. Dazu speichern Sie sich am besten, die beiden Werte vor einer Änderung, in 2 Variablen ab. Das könnte so aussehen:

A1=PEEK(V+17) und A2=PEEK(V+24)

Um nach der Graphikbearbeitung wieder in den Textmodus zu

gelangen, müssen Sie vorher die eben gespeicherten Variablen, in unserem Beispiel die Variablen A1 und A2, wieder in die Adressen V+17 und V+24 zurückschreiben. Dies geschieht mit:

POKE V+17,A1 und POKE V+24,A2

Nun sind Sie wieder in der Textseite, und können ganz normal weiterarbeiten.

Als nächstes müssen Sie die Graphikseite auf die richtige Farbe bringen. Hier müssen Sie jedes einzelne Byte ändern. Das geht natürlich am besten in einer FOR NEXT Schleife. Die erste Schleife setzt den Graphikbildschirm auf die gewünschte Farbe, während die zweite Schleife die Bildschirmseite, welche die einzelnen Punkte enthält, vollständig löscht.

An der Stelle ein paar Informationen über die Punkte.

Jeder Punkt wird als ein Bit dargestellt. Wie Sie wissen, kann ein Bit den Wert 0 oder 1 haben. 1 würde bedeuten, daß an dieser Stelle ein Punkt gesetzt ist. 0 bedeutet demnach, daß der Punkt nicht gesetzt ist. 8 Punkte zusammen ergeben ein Byte. In einer Zeile lassen sich 40 Byte darstellen. Dies bedeutet  $8 \times 40 = 320$  Punkte pro Zeile. Ähnlich verhält sich das mit den Spalten. In einer Spalte lassen sich 8 Punkte unterbringen. Bei 25 Zeilen ergibt das eine Auflösung von  $8 \times 25 = 200$  Punkten. Insgesamt stehen also 64.000 Punkte zur Verfügung.

Aber nun weiter. Betrachten wir einmal ein einzelnen Zeichen.  $8 \times 8$  Punkte pro Zeichen ergeben 64 Punkte in einem Zeichen. Jeder dieser Punkte läßt sich unabhängig von einem anderen Punkt setzen oder löschen. Nur bei den Farben gibt es Probleme. Im Farb RAM werden nicht die einzelnen Punkte, sondern die gesamte  $8 \times 8$  Matrix beachtet. Würde man für jeden einzelnen Punkt eine bestimmte Farbe wählen wollen, so müßte man alleine für den Farb RAM einen 64KB RAM großen Speicher benötigen. Das wäre ziemlich viel verlangt. Aus diesem Grund, müssen die 64 Punkte in diesem Zeichen die selbe Farbe haben. Das kommt auch der Darstellung auf dem Fernsehgerät zu gute. Ein höhere Farbauflösung würde zu einem völlig verschwommenen Bild führen.

Die Informationen sähen also folgendermaßen aus:

Ein Zeichen:     . . . . .  
                  . . . . .  
                  . . . . .  
                  . . . . .  
                  . . . . .  
                  . . . . .  
                  . . . . .  
                  . . . . .  
                  . . . . .  
                  . . . . .     = 64 Punkte (der selben Farbe)

Dieses Zeichen, mit den 64 Punkten, könnte zum Beispiel in der Speicherstelle 1024 ein Buchstabe sein, oder in der selben Speicherstelle die Position der entsprechenden Punkte im HI-RES Modus. Das klingt zunächst etwas kompliziert, ist es aber bei genauer Überlegung nicht.

Daraus erkennen Sie schon, daß es nicht möglich ist (zumindest ohne großen Programmieraufwand) Text und Graphik zu mischen. Das ist auf dem ersten Blick ein großer Nachteil. Leider ist es bei

keinem Computer dieser Preis- und Leistungsklasse so einfach möglich Text genauso wie Graphik zu behandeln. Dann würde der Computer keinen getrennten Speicher für Text und Graphik benötigen. Solche GRAPHIK-RECHNER liegen aber im Preis unvergleichbar über den Personal Computern. Und wer will schon aus seinem Wohnzimmer ein Rechenzentrum machen ?

Es gibt aber einen kleinen Trick, der es ermöglicht Graphiken zu beschriften. Dazu müssen Sie den Text, oder das einzelne Zeichen, zu einem Graphiksymbol machen. Hierzu lesen Sie das Zeichen aus der Zeichentabelle ein (s. Beispielsprogramm im Kapitel 2.4), und wandeln dieses Zeichen dann in ein Graphiksymbol um.

Nehmen wir aber an, daß sich an der Stelle tatsächlich ein Zeichen befindet. Gleichzeitig befindet sich jedoch an der Speicherstelle 1024 auch die Farbe des Graphiksymbols. Wie Sie ja wissen, hat das Graphiksymbol die selbe Größe wie ein normaler Buchstabe (8 x 8). Sind wir also einmal in der Graphikseite, und kehren danach wieder in die Textseite zurück, ohne in den Textmode gegangen zu sein, so sehen wir anstelle eines Buchstabens nur ein farbiges Kästchen. Würde der Commodore 64 zum Beispiel eine Fehlermeldung ausgeben, so würden statt dessen nur diese farbige Kästchen erscheinen, die nicht lesbar sind. Schaltet man aber wieder in den Textmodus, so können Sie einen Text wieder lesen - dafür verschwindet die Graphik.

Zum besseren Verständnis bringen wir Ihnen hier eine Tabelle der verschiedenen Speicherbereiche:

MODUS	BILDSCHIRMMRAM	FARBGRAM
Text	1024 - 2023	55296 - 56295
Graphik	8192 - 16383	1024 - 2023

Aus dieser Tabelle geht deutlich hervor, warum sich Text und Graphik nicht problemlos mischen lassen. Was für den Textmodus die Position des Zeichens ist, das ist im Graphikmodus die Farbe des Kästchens.

Ein weiteres Problem ist die Generierung der Punkte. Die zu jedem Zeichen (also auch zu einem Graphiksymbol) gehörende 8 x 8 Matrix, wird nämlich folgendermaßen gefüllt:

Der erste Punkt wird in die obere Reihe eines jeden Blockes gesetzt. Die Position des Punktes in dieser Reihe, wird durch den entsprechenden POKE Befehl gesteuert. Der nächste Punkt kommt in die zweite Reihe des selben Blockes usw. Nach 8 Reihen wird der Block gewechselt. Der nächste Block befindet sich rechts daneben bzw. am Anfang der nächsten Zeile. In einer Graphik sähe das so aus:

```

8192 -> . . . . .      8200 -> . . . . .
8193 -> . . . . .      8201 -> . . . . .
8194 -> . . . . .
8195 -> . . . . .      usw.
8196 -> . . . . .
8197 -> . . . . .
8198 -> . . . . .
8199 -> . . . . .

```

Nun müssen wir uns wieder etwas mit der Binärarithmetik

beschäftigen. Wir sagten bereits, daß die Lage des Punktes innerhalb der Speicherstelle abhängig ist von dem entsprechenden POKE. Was bedeutet das nun? Wenn wir uns den POKE Befehl einmal betrachten, sehen wir, daß er aus 2 Hälften besteht. Die erste Adresse gibt die Adresse an, die zweite Hälfte den Wert. Um also in der ersten Zeile des ersten Blockes auf dem Bildschirm einen Punkt zu setzen, müsste der erste Teil des POKE Befehls auf jeden Fall lauten: "POKE 8192,". Und hier beginnen dann die eigentlichen Schwierigkeiten. Ich kann ja mit diesem POKE nicht nur einen Punkt setzen, sondern maximal 8 Punkte gleichzeitig. Also muß man sich zunächst über die Lage des Punktes oder der Punkte im klaren sein. Wenn dann das Muster steht, muß es noch in binäre Form übersetzt werden. Das sieht dann so aus:

#### BEISPIEL:

Der linke und der rechte Punkt in der Speicherstelle 8192 sollen eingeschaltet werden. Das bedeutet, daß das linke (b7) Bit und das rechte Bit (b0) gesetzt werden müssen:

1 0 0 0 0 0 0 1

Da man aber nicht sagen kann "POKE 8192,10000001", muß die binäre Zahl noch in dezimale Form übersetzt werden. Wir erinnern uns: b7 entspricht  $2^7$  (=128) und b0 bedeutet  $2^0$  (=1). Also ist das Ergebnis 129 oder zusammen: "POKE 8192,129". Nun befinden sich in der ersten Zeile 2 Punkte (sofern der Graphikmodus eingeschaltet war). Achten Sie also bei der Programmierung von Graphik stets auf diese Besonderheit des Commodore 64. Es könnte Ihnen sonst passieren, daß Sie sich genauso wundern, wie wir es taten, als wir zum ersten mal mit der Graphik experimentierten.

Auf den nächsten Seiten folgt nun das schon oben erwähnte Programm HI-RES GRAPHIK-HILFE, das Ihnen die Programmierung von Graphik erheblich vereinfachen kann.

Noch ein Hinweis zum Schluß: Wer eine sehr komfortable Unterstützung der Graphikprogrammierung sucht, dem kann das Programm SUPERGRAPHIK 64 - aus dem Hause DATA BECKER - empfohlen werden. Dieses schnelle Maschinenprogramm, das keinen BASIC Speicherplatz belegt, bringt auf dem Commodore alle Befehle des bekannten SUPERGRAPHIC Boards für den Commodore 64 und dazu noch spezielle Befehle. An diesem Beispiel zeigt sich, wie überlegen der Commodore 64 selbst seinen größeren Brüdern gegenüber ist.

Während in den großen Commodore Computern erst hardwaremäßig die Voraussetzungen für hochauflösende Graphik geschaffen werden mußten, genügt beim Commodore 64 bereits ein entsprechendes Programm.

Auf der nächsten Seite zeigen wir Ihnen den Befehlsvorrat des SUPERGRAPHIK 64 Programmes. Damit wird die Programmierung von Graphik zum wahren Vergnügen.

### \*\*\* SUPERGRAPHIK/64 \*\*\*

SUPERGRAPHIK/64 ist eine komfortable Befehlserweiterung für den Commodore 64, die die Programmierung der hochauflösenden Farbgrafik in BASIC möglich macht. SUPERGRAPHIK/64 bietet als Befehle

!GCLEAR	- Löscht den Grafikbildschirm
!GRAPHICS	- Wählt eine der sechs Betriebsarten
!DOT	- Setzt einen Punkt an die angesprochene Stelle
!CDOT	- Löscht einen Punkt
!TEST	- Testet ob ein Punkt gesetzt
!LINE	- Zeichnet eine durchgezogene Linie
!CLINE	- Löscht eine Linie
!DLINE	- Zeichnet eine gepunktete Linie
!FILL	- Füllt ein ganzes Feld aus
!CLEAR	- Löscht ein ganzes Feld
!FRAME	- Zeichnet einen Rahmen
!CFRAME	- Löscht einen Rahmen
!MOVE	- Bewegt den Graphik-Cursor
!DRAW	- Zeichnet eine Linie ab Cursor-Position
!BSAV	- Speichert den Graphikbildschirm auf Kassette / Disk
!RECALL	- Überträgt den Graphikbildschirm von Kassette / Disk in den Arbeitsspeicher
!DISP	- Zeichnet Sonderzeichen
!HARD	- Druckt den Graphikbildschirm aus
!TRANS	- Wechselt zwischen den beiden Graphikbildschirmen
!SPRITE	- Definiert ein Sprite
!SMODE	- Wählt Farbe und Größe eines Sprites
!SMOVE	- Bewegt einen Sprite
!BCOL	- Gibt die Hintergrundfarbe an
!LCOL	- Gibt die Linienfarbe an
!TEXT	- Schreibt einen Text in den Graphikbildschirm
!CIRCLE	- Zeichnet einen Kreis
!DCIRCLE	- Zeichnet einen gepunkteten Kreis
!CCIRCLE	- Löscht einen Kreis

SUPERGRAPHIK 64 wird einfach von Kassette oder Diskette geladen und ist dann sofort einsatzfähig.

Erschließen Sie sich die faszinierende Welt graphischer Datenverarbeitung mit dem Commodore 64 und SUPERGRAPHIK/64.

(c) Copyright by DATA BECKER 1983

# Hier ist nun das Assemblerlisting zum GRAFIK AID-Programm

```

110: c000      cr      = 13      : carriage return
120: c000      esc     = 27      ; escape
130: c000      xcoord  = $14      ;
140: c000      flag    = $97      ; punkt setzen/loschen
150: c000      maske   = flag     ; maske für hardcopy
160: c000      sa      = $b9      ; sekundaradresse
170: c000      tap     = $fd
180: c000      adr     = tap
190: c000      av      = tap
200: c000      code    = xcoord
210: c000      spalte  = code + 1
220: c000      collo   = $400     ; anfang hi-res farbrau
230: c000      colhi   = $800     ; ende "
240: c000      gralo   = $2000    ; anfang hi-res bitmap
250: c000      grahi   = $4000    ; ende "
260: c000      getcor  = $b7eb    ; holt x- und y-coordinaten
270: c000      chicco  = $ae4d    ; pruit auf yaaa
280: c000      getbyt  = $b79e    ; holt byte-wert
290: c000      video   = $d000    ; videocontroller
300: c000      getpar  = $e1d4    ; holt filenames und geratenummer
310: c000      clrscr  = $e54a    ; löscht bildschirm
320: c000      chlout  = $ffc9    ; setzt ausgabegerat
330: c000      clrch   = $ffcc    ; ausgabe auf default
335: c000      print   = $ffd2    ; ausgabe routine
340: c000      load    = $ffd5    ; load routine
350: c000      save    = $ffd8    ; save routine
      ;
370: c000      jmp     = $c000    ; sprungtabelle für funktionen
380: c000 4c 1e c0    jmp init    ; grafik-modus einschalten
390: c003 4c 3c c0    jmp clear   ; grafik löschen
400: c006 4c 51 c0    jmp color   ; farbe setzen
410: c009 4c 6d c0    jmp revers  ; grafik invertieren
420: c00c 4c 89 c0    jmp set     ; grafikpunkt setzen
430: c00f 4c 86 c0    jmp reset   ; grafikpunkt löschen
440: c012 4c 52 c1    jmp gload   ; grafik laden
450: c015 4c 39 c1    jmp gsave   ; grafik abspeichern
460: c018 4c 75 c1    jmp hard    ; hardcopy
470: c01b 4c 61 c1    jmp goff    ; grafik aus
      ;
490: c01e 20 3c c0    jsr init    ; grafik löschen
500: c021 ad 11 d0    lda video+17 ; grafik einschalten
510: c024 8d 70 c1    sta store1
520: c027 ad 18 d0    lda video+24
530: c02a 8d 71 c1    sta store2
540: c02d a9 3b      lda $27+$32
550: c02f 8d 11 d0    sta video+17
560: c032 a9 18      lda $16+$8
570: c034 8d 18 d0    sta video+24
580: c037 a2 10      ldx $10    ; punktfarbe weiss,
590: c039 4c 57 c0    jmp col    ; hintergrundfarbe schwarz
      ;
610: c03c a0 00      ldy #0      ; grafik speicher löschen
620: c03e a2 20      ldx #>gralo
630: c040 84 fd      sty tmp
640: c042 86 fe      stx tmp+1
650: c044 98         tya
650: c045 ea         nop
660: c046 91 fd      clr      sta (tmp),y

```

```

670: c048 c8          iny
680: c049 d0 fb          bne clr
690: c04b eb fe          inc tmp+1
700: c04d ca              dex          ; nachste page
710: c04e d0 fb          bne clr
720: c050 60              rts

;
740: c051 20 fd ae color jsr chkcoa ; farbe setzen
750: c054 20 9e b7        jsr getbyt ; farbcodes holen
760: c057 a0 00          ldy #0
770: c059 a9 04          lda #0> collo
780: c05b 84 fd          sty tmp
790: c05d 85 fe          sta tmp+1
800: c05f 8a              txa          ; farbcodes
810: c060 a2 04          ldx #0> colhi-collo ; anzahl pages
820: c062 91 fd          sta (tmp),y
830: c064 c8              iny
840: c065 d0 fb          bne coli
850: c067 eb fe          inc tmp+1
860: c069 ca              dex          ; nachste page
870: c06a d0 fb          bne coli
880: c06c 60              rts

;
900: c06d a6 00          revers ldy #0          ; grafik invertieren
910: c06f a9 20          lda #0>gralo
920: c071 84 fd          sty tmp
930: c073 85 fe          sta tmp+1
940: c075 a2 20          ldx #0>grahi-gralo ; anzahl pages
950: c077 b1 fd          revl  lda (tmp),y
960: c079 49 ff          eor #$ff ; alle bits umdrehen
970: c07b 91 fd          sta (tmp),y
980: c07d c8              iny
990: c07e d0 f7          bne revl
1000: c080 eb fe          inc tmp+1
1010: c082 ca              dex          ; nachste page
1020: c083 d0 f2          bne revl
1030: c085 60              rts          ; fehlerhafte koordinaten bei set

;
1050: c086 a9 80          reset  lda #$80          ; grafikpunkt löschen
1060: c088 2c              .byt $2c
1070: c089 a9 00          set     lda #0          ; grafikpunkt setzen
1080: c08b 85 97          sta flag
1090: c08d 20 fd ae        jsr chkcoa ; komma
1100: c090 20 eb b7        jsr getcor ; x nach xcoord, y nach x-register
1110: c093 e0 c8          cpx #200
1120: c095 b0 ee          bcs ill          ; y-coordinate > 199 ?
1130: c097 a5 15          lda xcoord+1
1140: c099 c9 01          cmp #320
1150: c09b 90 08          bcc ok
1160: c09d d0 e6          bne ill
1170: c09f a5 14          lda xcoord
1180: c0a1 c9 40          cmp #320 ; x-coordinate > 319 ?
1190: c0a3 b0 e0          bcs ill
1200: c0a5 8a              ok     txa          ; y-coordinate in akku
1210: c0a6 4a              lsr
1210: c0a7 4a              lsr
1210: c0a8 4a              lsr
1210: c0a9 0a              asl          ; durch 8 mal 2
1220: c0aa a8              tay
; offy = 320 * int(y/8) + (y and 7)

```

```

1240: c0ab b9 ff c0      lda mult,y
1250: c0ae 8d 73 c1      sta offy
1260: c0b1 b9 00 c1      lda mult+1,y ; mal 320
1270: c0b4 8d 74 c1      sta offy+1
1280: c0b7 8a           txa           ; v-coordinate
1290: c0b8 29 07      and #7
1300: c0ba 18           clc
1310: c0bb 8d 73 c1      adc offy
1320: c0be 8d 73 c1      sta offy
; offx = 8 * int(x/8)
1340: c0c1 a5 14      lda xcoord
1350: c0c3 29 f8      and %111111000
1360: c0c5 8d 72 c1      sta offx
1370: c0c8 18           clc           ; av = gralo + offy * offx
1380: c0c9 a9 00      lda #<gralo
1390: c0cb 6d 73 c1      adc offy
1400: c0ce 85 fd      sta av
1410: c0d0 a9 20      lda #>gralo
1420: c0d2 6d 74 c1      adc offy+1
1430: c0d5 85 fe      sta av+1
1440: c0d7 18           clc
1450: c0d8 a5 fd      lda av
1460: c0da 6d 72 c1      adc offx
1470: c0dd 85 fd      sta av
1480: c0df a5 fe      lda av+1
1490: c0e1 a5 15      adc xcoord+1
1500: c0e3 85 fe      sta av+1
; ma = 2^((7-x)and7)
1520: c0e5 a5 14      lda xcoord
1530: c0e7 29 07      and #7
1540: c0e9 49 07      eor #7
1550: c0eb 8a           tax
1560: c0ec 8d 31 c1      lda grbit,x ; 2 hoch ma aus tabelle
1570: c0ef a0 00      ldy #0
1580: c0f1 24 97      bit flag
1590: c0f3 10 05      bpl set1
1600: c0f5 89 ff      eor %fff
1610: c0f7 31 fd      and (av),y ; punkt loschen
1620: c0f9 2c          .byt $2c
1630: c0fa 11 fd      set1 ora (av),y ; punkt setzen
1640: c0fc 91 fd      sta (av),y
1650: c0fe 60          rts
;
; multiplikationstabelle n*320, n=0,24
;
1675: c0ff          mult      = *
1680: c0ff 00 00 40 01 80 02 c0 03
1685: c107 00 05 40 06 80 07 c0 08
1690: c10f 00 0a 40 0b 80 0c c0 0d
1695: c117 00 0f 40 10 80 11 c0 12
1700: c11f 00 14 40 15 80 16 c0 17
1705: c127 00 19 40 1a 80 1b c0 1c
1710: c12f 00 1e
;
1720: c131 01 02 04 grbit .byt 1,2,4,8,$10,$20,$40,$80 ; zweierpotenzen
c134 08 10 20
c137 40 80
;
1740: c139 20 fd ae gsave jsr chkcom ; grafik abspeichern
1750: c13c 20 d4 e1 jsr getpar ; filenames und gerateadresse holen

```



```

1760: c13f a2 00      ldx  #< grahi
1770: c141 a0 40      ldy  #> grahi
1780: c143 a9 00      lda  #< gralo
1790: c145 B5 fd      sta  tmp
1800: c147 a9 20      lda  #> gralo
1810: c149 B5 fe      sta  tmp+1
1820: c14b a9 fd      lda  #tmp
1830: c14d B5 b9      sta  sa      ; absolut speichern
1840: c14f 4c dB ff   jmp  save

;
1860: c152 20 fd ae gload  jsr  chkcom ; grafik laden
1865: c155 20 d4 e1      jsr  getpar ; filenames und gerateadresse holen
1870: c158 a9 01      lda  #1      ; absolut laden
1880: c15a B5 b9      sta  sa
1890: c15c a9 00      lda  #0      ; load flag
1900: c15e 4c d5 ff   jmp  load

;
1920: c161 ad 70 c1 goff  lda  store1 ; grafik aus
1930: c164 Bd 11 d0      sta  video+17
1940: c167 ad 71 c1      lda  store2
1950: c16a Bd 18 d0      sta  video+24
1960: c16d 4c 44 e5   jmp  clrscr ; bildschirm löschen

;
1980: c171      store1  +=  ++1
1990: c172      store2  +=  ++1
2000: c173      offx    +=  ++1
2010: c175      offy    +=  ++2

;
; hardcopy fur
; epson fx 80 mit vc-interface
;
2060: c175 20 fd ae hard  jsr  chkcom ; komma
2070: c178 20 9e b7      jsr  getbyt ; logische filenummer
2080: c17b 20 c9 ff      jsr  chkout ; ausgabe auf drucker
2090: c17e a9 00      lda  #<gralo
2100: c180 a0 20      ldy  #>gralo
2110: c182 B5 fd      sta  adr      ; zeiger auf grafik
2120: c184 B4 fe      sty  adr+1
2130: c186 a2 19      ldx  #25      ; anzahl der zeilen
2140: c188 a0 07      ldy  #7
2150: c18a 2c      .byt $2c
2160: c18b a0 05      ldy  #5
2170: c18d b9 d6 c1 gad   lda  gad,y
2180: c190 20 d2 ff      jsr  print ; drucker auf grafik-modus
2190: c193 B8      dey
2200: c194 10 f7      bpl  gad
2210: c196 a9 28      lda  #40
2220: c198 B5 15      sta  spalte
2230: c19a a9 B0      lda  #80
2240: c19c B5 97      sta  maske
2250: c19e a9 00      lda  #0
2260: c1a0 B5 14      sta  code
2270: c1a2 a0 07      ldy  #7
2280: c1a4 b1 fd      lda  (adr),y ; bitmuster zusammen setzen
2290: c1a6 25 97      and  maske
2300: c1a8 f0 07      beq  tt2      ; bit nicht gesetzt
2310: c1aa a5 14      lda  code
2320: c1ac 19 de c1      ora  gbit,y ; bit gesetzt
2330: c1af B5 14      sta  code
2340: c1b1 B8      dey

```

```

2360: c1b2 10 f0          bpl bits
2370: c1b4 a5 14          lda code
2380: c1b6 20 d2 ff       jsr print ; code an drucker
2390: c1b9 46 97          lsr maske
2400: c1bb 90 e1          bcc bytes
2410: c1bd a5 fd          lda adr
2420: c1bf 69 07          adc #7 ; plus carry = 8
2430: c1c1 85 fd          sta adr ; adresse erhöhen
2440: c1c3 90 02          bcc tt?
2450: c1c5 e6 fe          inc adr+1
2460: c1c7 c6 15          tt? dec spalte ; nachste spalte
2470: c1c9 d0 cf          bne spalten
2480: c1cb ca             dex ; nachste zeile
2490: c1cd d0 bd          bne zeilen
2500: c1ce a9 0d          lda #cr
2510: c1d0 20 d2 ff       jsr print
2520: c1d3 4c cc ff       jmp clrch ; ausgabe wieder auf bildschirm
2530: c1d6 01 40          gmod .bvt 320, 320 ; anzahl grafikpunkte
2540: c1d8 06 2a 1b       .bvt 6, "*", esc, cr ; grafik modus
2550: c1db 0d
2560: c1de 31 1b          .bvt "l", esc ; 8 punkte pro zeile
2570: c1e0 80 40 20 gbit .bvt #80, #40, #20, #10, 8, 4, 2, 1 ; zweierpotenzen
2580: c1e1 10 08 04
2590: c1e4 02 01
2600: c1e6               .end

```

Hier wieder ein Ladeprogramm in BASIC

```

100 for i = 49152 to 49637
110 read x : poke i, x : s=s+x : next
120 data 76, 30, 192, 76, 60, 192, 76, 81, 192, 76, 109, 192
130 data 76, 137, 192, 76, 134, 192, 76, 82, 193, 76, 57, 193
140 data 76, 117, 193, 76, 97, 193, 32, 60, 192, 173, 17, 208
150 data 141, 112, 193, 173, 24, 208, 141, 113, 193, 169, 59, 141
160 data 17, 208, 169, 24, 141, 24, 208, 162, 16, 76, 87, 192
170 data 160, 0, 162, 32, 132, 253, 134, 254, 152, 234, 145, 253
180 data 200, 208, 251, 230, 254, 202, 208, 246, 96, 32, 253, 174
190 data 32, 158, 183, 160, 0, 169, 4, 132, 253, 133, 254, 138
200 data 162, 4, 145, 253, 200, 208, 251, 230, 254, 202, 208, 246
210 data 96, 160, 0, 169, 32, 132, 253, 133, 254, 162, 32, 177
220 data 253, 73, 255, 145, 253, 200, 208, 247, 230, 254, 202, 208
230 data 242, 96, 169, 128, 44, 169, 0, 133, 151, 32, 253, 174
240 data 32, 235, 183, 224, 200, 176, 238, 165, 21, 201, 1, 144
250 data 8, 208, 230, 165, 20, 201, 64, 176, 224, 138, 74, 74
260 data 74, 10, 168, 185, 255, 192, 141, 115, 193, 185, 0, 193
270 data 141, 116, 193, 138, 41, 7, 24, 109, 115, 193, 141, 115
280 data 193, 165, 20, 41, 248, 141, 114, 193, 24, 169, 0, 109
290 data 115, 193, 133, 253, 169, 32, 109, 116, 193, 133, 254, 24
300 data 165, 253, 109, 114, 193, 133, 253, 165, 254, 101, 21, 133
310 data 254, 165, 20, 41, 7, 73, 7, 170, 189, 49, 193, 160
320 data 0, 36, 151, 16, 5, 73, 255, 49, 253, 44, 17, 253
330 data 145, 253, 96, 0, 0, 64, 1, 128, 2, 192, 3, 0
340 data 5, 64, 6, 128, 7, 192, 8, 0, 10, 64, 11, 128
350 data 12, 192, 13, 0, 15, 64, 16, 128, 17, 192, 18, 0
360 data 20, 64, 21, 128, 22, 192, 23, 0, 25, 64, 26, 128
370 data 27, 192, 28, 0, 30, 1, 2, 4, 8, 16, 32, 64
380 data 128, 32, 253, 174, 32, 212, 225, 162, 0, 160, 64, 169
390 data 0, 133, 253, 169, 32, 133, 254, 169, 253, 133, 185, 76
400 data 216, 255, 32, 253, 174, 32, 212, 225, 169, 1, 133, 185
410 data 169, 0, 76, 213, 255, 173, 112, 193, 141, 17, 208, 173

```

```

420 data 113,193,141, 24,208, 76, 68,229, 0, 0, 0, 0
430 data 0, 32,253,174, 32,158,183, 32,201,255,169, 0
440 data 160, 32,133,253,132,254,162, 25,160, 7, 44,160
450 data 5,185,214,193, 32,210,255,136, 16,247,169, 40
460 data 133, 21,169,128,133,151,169, 0,133, 20,160, 7
470 data 177,253, 37,151,240, 7,165, 20, 25,222,193,133
480 data 20,136,208,240,165, 20, 32,210,255, 70,151,144
490 data 225,165,253,105, 7,133,253,144, 2,230,254,198
500 data 21,208,207,202,208,189,169, 13, 32,210,255, 76
510 data 204,255, 1, 64, 6, 42, 27, 13, 49, 27,128, 64
520 data 32, 16, 8, 4, 2, 1
530 if s <> 60459 then print "fehler in datas !!" : end
540 print "ok !"

```

Wie wendet man nun das Grafik-Hilfsprogramm an ?

Der Aufruf der einzelnen Funktionen geschieht über SYS-Aufrufe, wobei teilweise noch Parameter mit übergeben werden. Zu Beginn des Programms werden die Adressen der Routinen an Variablen übergeben, die beim späteren Aufruf verwendet werden. Danach können noch - soweit erforderlich - Parameter durch Komma getrennt übergeben werden. In unserem Beispiel wurden folgende Namen verwendet:

```

X - horizontale Koordinate eines Grafikpunkts, 0 bis 319
Y - vertikale Koordinate, 0 bis 199
   Die Koordinate 0,0 liegt in der linken oberen Ecke
PF- Farbe eines Grafikpunkts, 0 bis 15
HF- Farbe des Hintergrunds, 0 bis 15
LF- logische Filenummer des Druckers, 1 bis 255

```

```

100 IN = 12*4096: REM SYS IN - GRAFIK EINSCHALTEN
110 CL = IN+3 : REM SYS CL - GRAFIK LoSCHEN
120 CO = IN+6 : REM SYS CO,PF+16+HF - COLOR SETZEN
120 RV = IN+9 : REM SYS RV - GRAFIK INVERTIEREN
130 SE = IN+12 : REM SYS SE,X,Y - GRAFIKPUNKT SETZEN
140 RS = IN+15 : REM SYS RS,X,Y - GRAFIKPUNKT LoSCHEN
150 GL = IN+18 : REM SYS GL,"NAME",1 ODER 8 - GRAFIK LADEN
160 GS = IN+21 : REM SYS GS,"NAME",1 ODER 8 - GRAFIK ABSPEICHERN
170 HD = IN+24 : REM SYS HD,LF - HARDCOPY AUF DRUCKER
180 OF = IN+27 : REM SYS OF - GRAFIK ABSCHALTEN
200 SYS IN : REM GRAFIK EIN
210 PF = 1 : REM PUNKTFARBE = WEISS
220 HF = 0 : REM HINTERGRUNDFARBE = SCHWARZ
230 SYS CO, 16+PF+HF : REM FARBE SETZEN
240 REM ZEICHNEN DER X-ACHSE
250 FOR X = 0 TO 319 : SYS SE, X, 100 : NEXT
260 REM ZEICHNEN DER Y-ACHSE
270 FOR Y = 0 TO 199 : SYS SE, 160, Y : NEXT
280 REM ZEICHNEN DER SINUS-KURVE
290 PI = 3.14159265 : X = 0
300 FOR I = -PI TO PI STEP 2*PI/319
310 SYS SE, X, 100*99*SIN(I)
320 X = X+1 : NEXT
330 SYS GS, "SINUSKURVE",8 : REM GRAFIK AUF DISKETTE ABSPEICHERN
340 OPEN 1,4,1 : REM DRUCKER OFFNEN, EPSON-MODUS
350 SYS HD,1 : REM HARDCOPY AUF DRUCKER GEBEN
360 CLOSE 1 : REM DRUCKERFILE WIEDER SCHLIESSEN
370 SYS OF : REM GRAFIK AUSSCHALTEN

```

### 3.7 Sprites - Das Zauberwort des Commodore 64

#### EINFÜHRUNG

Neben der hochauflösenden Graphik ist mit Sicherheit die Erzeugung sogenannter SPRITES das hervorragende Merkmal des Commodore 64. Diese Sprites sind eigenständige, kleine Graphiken, die unabhängig voneinander kontrolliert werden können. Gleichzeitig ersparen Sie sich bei der Verwendung von Sprites, das ewige Umschalten auf die Graphikseite, da der Commodore 64 diese Sprites nach einem anderen Prinzip, als die normale Graphik verwaltet.

#### MÖGLICHKEITEN

Der Commodore 64 hat die Möglichkeit, bis zu 8 Sprites (von 0 bis 7) auf dem Bildschirm darzustellen. Sie brauchen einfach einen bestimmten Speicherbereich zu wählen, dem entsprechenden Sprite eine Nummer zu geben, und bei Bedarf ein oder auszuschalten. Durch Angabe der X-Y Koordinaten, können Sie das Sprite von einer Seite des Bildschirms zur anderen bewegen, ohne die vorherige Position des Sprites löschen zu müssen. Sie brauchen keinen Bildschirm RAM mehr zu verändern oder irgendwelche Veränderungen im Farb RAM vorzunehmen. Das geschieht nun alles automatisch (bzw. vom 6569 Video Display Chip gesteuert).

Außerdem können Sie die Sprites in vertikaler und / oder horizontaler Richtung vergrößern; eine Kollision zwischen verschiedenen Sprites und einem bestimmten Hintergrund feststellen und darauf reagieren; die Priorität wählen - Sprites können vor oder hinter dem Hintergrund erscheinen. Dies ermöglicht auch eine räumliche Darstellung der Sprites. Der Commodore 64 hat also die Möglichkeit, dreidimensionale Graphiken zu erzeugen. Durch dieses Überlagern von Sprites, lassen sich Effekte erzeugen, die man bisher nur mit großen Schwierigkeiten, und enormen Programmieraufwand erstellen konnte.

#### AUFBAU

Bei der Benutzung der Sprites, ist es von besonderer Wichtigkeit, über die Binärarithmetik und die Register des Graphikprozessors 6569 Bescheid zu wissen (Kapitel 1.1 und 3.1.1). Falls Sie diese Kapitel noch nicht gelesen haben sollten, oder diese noch nicht richtig verstanden haben, lesen Sie sich die entsprechenden Abschnitte am besten noch einmal durch. Dadurch werden Sie die nächsten Seiten wesentlich leichter verstehen.

Die wichtigste Aufgabe bei der Generierung von Sprites, fällt den 46 Registern zu (siehe Kapitel 3.1.1). Mit diesen Registern lassen sich Bewegung, Farbe und alle anderen Besonderheiten der Sprites, steuern. Jedes Register besteht aus 8 Bit, die der Anwender nach seinen Erfordernissen setzen oder löschen kann. Ein anderer wichtiger Teil für die Sprites, ist die eigentliche Position der Punkte. Wie Sie wissen, besteht jedes Sprite aus 24 x 21 Punkten, die entsprechend der Graphikprogrammierung gesetzt werden können. Da man aber durch einen POKE Befehl maximal 8 Punkte setzen oder löschen kann, aber in der horizontalen Ausdehnung des Sprites, insgesamt 24 Punkte zur Verfügung stehen, mußte man einen besonderen Weg gehen.

So setzt sich jedes Sprite aus 3 SERIEN zusammen, die jeweils

aus einer 3 x 21 Punktematrix bestehen. Bei der Programmierung, ist nun darauf zu achten, daß jeweils die 3 Serien einer Zeile immer nebeneinander stehen.

Hier die Veranschaulichung über den Aufbau einer Serie:

		SERIE												
		1				2				3				
ZEILE	1	.	.	.	.	.	.	.	.	.	.	.	.	.
ZEILE	2	.	.	.	.	.	.	.	.	.	.	.	.	.
ZEILE	3	.	.	.	.	.	.	.	.	.	.	.	.	.
ZEILE	4	.	.	.	.	.	.	.	.	.	.	.	.	.
ZEILE	5	.	.	.	.	.	.	.	.	.	.	.	.	.
ZEILE	6	.	.	.	.	.	.	.	.	.	.	.	.	.
:		.						.					.	
:		.						.					.	
ZEILE	21	.	.	.	.	.	.	.	.	.	.	.	.	.

Jede einzelne dieser Zeilen, setzt sich also aus 3 Serien, mit je 8 Bit zusammen. Der Informationsinhalt eines jeden Sprites kann sich aus zwei Möglichkeiten zusammensetzen.

### 1.) Das einfarbige Sprite

Bei diesem Sprite, entspricht jedes Bit genau einem Punkt. Dieser Punkt hat dann entweder die Farbe, die in seinem entsprechenden Register vermerkt ist, oder aber die Farbe des Hintergrundes - das heißt, daß dieser Punkt als nicht gesetzt erscheint.

### 2.9 Das MULTICOLOR Sprite

Bei diesem Sprite entsprechen jeweils zwei Bit einem Punkt. Diese zwei Bits geben Auskunft über das Register, aus dem die Farbe des Sprites entnommen wird. So läßt sich ein Multicolr Sprite, aus maximal drei Farben zusammensetzen (außerdem die Farbe des Hintergrundes für einen nicht gesetzten Punkt).

Auf die Programmierung der Sprite gehen wir im nächsten Kapitel noch genauer ein.

### 3.7.1 Programmierung der Sprites

#### EINFÜHRUNG

Man geht bei der Programmierung der Sprites genauso vor, wie wir es für die Programmierung in Maschinensprache gelernt haben.

Man überlegt sich also zuerst die Position der Punkte, die gesetzt werden sollen, wandelt dieses Binärmuster dann in Dezimalzahlen um, und hat damit die entsprechenden Werte für die POKE Befehle. Da man diese Punkte aber nicht immer wieder neu zu berechnen braucht, also ganz anders, als bei der Graphikprogrammierung, hat man hier die Möglichkeit, die Werte in DATA Zeilen bereit zu stellen. Dann liest man diese Werte, mit Hilfe einer FOR NEXT Schleife ein, schreibt diese in einen bestimmten Speicherbereich, und schon steht einem das somit erstellte Sprite zur Verfügung. Wichtig ist nur, daran zu denken, daß auch nicht gesetzte Punkte, in Form einer 0 an der entsprechenden Stelle, mit anzugeben. Um die Programmierung deutlicher zu gestalten, sollten Ihre DATA Zeilen so aussehen:

```
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 234,150,080
1120 DATA 194,170,168
1130 DATA 192,170,168
1140 DATA 000,032,128
1150 DATA 000,170,160
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

Sie können an diesem Aufbau deutlich die 3 x 21 Struktur der Spriteprogrammierung erkennen. Jede Zahl zwischen 1 und 255 repräsentiert ein bestimmtes Punktemuster innerhalb eines Sprites. So ist es also möglich, jede beliebige Figur, von einem Punkt bis hin zum vollen 24 x 21 Block, zu programmieren. Worin liegt aber nun der Unterschied zwischen der hochauflösenden Graphik und den Sprites?

#### ANWENDUNG

Um diese Frage zu beantworten, muß man zunächst einmal das Anwendungsgebiet der Sprites betrachten: Es sind hauptsächlich Spiele, und verschiedene Trickdarstellungen, die diese Anwendung interessant machen. Denn im Gegensatz zu einer Graphik, bleiben hier die Formen der Figuren immer gleich. Es kann höchstens vorkommen, daß sie sich ausdehnen. Aber ihre Struktur bleibt immer gleich. Diese Figuren müssen sich aber bewegen können. Graphiken dagegen, haben keine feste Formen - man kann zum

Beispiel an das Plotten einer Funktion denken. Oder stellen wir uns einmal die bekannte "Kuchengraphik" vor, wie sie zum Beispiel bei der Darstellung von Wahlergebnissen vorkommen, oder die graphische Darstellung eines Geschäftsjahres. Der Commodore 64 ist in der Lage beide Formen der Darstellung zu ermöglichen.

Die Sprites lassen sich aber ebenso gut auch für gewerbliche Zwecke einsetzen. Möglichkeiten hierfür, bietet zum Beispiel die Erzeugung von Laufschrift. Stellen Sie sich einmal eine Repräsentation eines Produktes, mit Hilfe des Commodore 64, seiner Graphik, seiner Farbe und seines Tones vor !

## DIE IDEE DER PROGRAMMIERUNG

Jetzt aber zur eigentlichen Programmierung der Sprites.

Noch einmal zu unserem Beispiel von oben. Was stand da eigentlich in den DATA Zeilen ? Um dieses zu erkennen, müssen wir die Dezimalzahlen in Binärzahlen umwandeln:

### SERIE

	1	2	3
ZEILE 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 2	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 3	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 4	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 5	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 6	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 7	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 8	0 0 0 0 0 0 0 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
ZEILE 9	0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0
ZEILE 10	1 1 0 0 0 0 0 0	1 0 1 0 1 0 1 0	1 0 0 0 0 0 0 0
ZEILE 11	1 1 0 0 0 0 1 0	1 0 0 1 0 1 1 0	0 1 0 1 0 0 0 0
ZEILE 12	1 1 1 0 1 0 1 0	1 0 0 1 0 1 1 0	0 1 0 1 0 0 0 0
ZEILE 13	1 1 0 0 0 0 1 0	1 0 1 0 1 0 1 0	1 0 1 0 1 0 0 0
ZEILE 14	1 1 0 0 0 0 0 0	1 0 1 0 1 0 1 0	1 0 1 0 1 0 0 0
ZEILE 15	0 0 0 0 0 0 0 0	0 0 1 0 0 0 0 0	1 0 0 0 0 0 0 0
ZEILE 16	0 0 0 0 0 0 0 0	1 0 1 0 1 0 1 0	1 0 1 0 0 0 0 0
ZEILE 17	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 18	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 19	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 20	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
ZEILE 21	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

Auch wenn Sie es nicht genau erkennen können, hier wurde ein Hubschrauber programmiert. Auf dem Papier kann man dieses Sprite natürlich nicht so darstellen, wie es auf dem Fernseher möglich ist.

### EINSCHALTEN

Nach der Generierung der Sprites, besteht nun die Möglichkeit diese beliebig zu verändern.

Als erstes wird dazu das Sprite eingeschaltet. Hierfür wird in das Register 21 die Kennziffer der einzuschaltenden Sprites gebracht.

Sprite: 7 6 5 4 3 2 1 0

Bit:        b7    b6    b5    b4    b3    b2    b1    b0

Durch POKE 21,1 wird also das Sprite 0 eingeschaltet, POKE 21,3 schaltet demnach Sprite 0 und Sprite 1 ein, und POKE 21,255 würde alle Sprites aktivieren.

#### SPICHERBEREICH

Danach muß die Adresse des Sprites definiert werden. Dazu stehen die Adressen 2040 bis 2047 zur Verfügung. In diesen Adressen wird der Anfangsbereich der Sprites festgelegt. Ein Sprite besteht aus 63 Byte. Durch die Angabe der Blockzahl, in der zu dem Sprite gehörenden Adresse, wird der entsprechende Speicherbereich definiert. Als Beispiel soll nun Block 13 gelten.  $13 * 64 \text{ Byte}$  (Länge eines Sprites + 1, für die Startadresse des nächsten Sprites) ergibt die Anfangsadresse 832.

Register Pointer		Adresse
11	* 64 =	704
13	* 64 =	832
14	* 64 =	896
15	* 64 =	960

Mit diesen Blöcken können Sie aber nur 4 verschiedene Sprites adressieren. Um auch noch andere Sprites ansprechen zu können, müssen Sie zunächst den BASIC Start verschieben, um so genügend Raum für andere Sprites zu schaffen. Diese Verschiebung können Sie folgendermaßen bewerkstelligen:

```
POKE 44,10      (Basic Start an $0A00)
POKE 10*256,0   (Erstes BASIC Byte = 0)
NEW             (Zurücksetzen der Pointer)
```

In der folgenden Tabelle finden Sie die Adressen für die Register Pointer der entsprechenden Sprites:

Adresse:	2040	2041	2042	2043	2044	2045	2046	2047
Sprite:	0	1	2	3	4	5	6	7

Wenn man nun in die Adresse 2040 den Wert 13 schreibt, so bedeutet das also, daß das Sprite ab Speicherstelle 832 zu finden ist. Nun kann ich auch an die Stelle 2041 den Wert 13 schreiben, was bedeuten würde, daß Sprite 1 und Sprite 2 in der Form identisch wären. Alle anderen Informationen, wie Position, Farbe etc. können dagegen unterschiedlich sein, was natürlich den Programmieraufwand auf ein Minimum reduziert. Wie Sie sehen, braucht bei der Programmierung keine weiteren Adressen verändert zu werden. Es wird einfach die Nummer des Sprites angegeben. Der Computer erledigt den Rest. So wird die Programmierung der Sprites zu einem Kinderspiel.

Nun werden durch eine FOR NEXT Schleife alle Werte des Sprites in den Speicherbereich ab 832 übertragen.

```
FOR I=0 TO 62: REM 63 BYTES EINES SPRITES
  READ X: REM LESEN DES BYTES
  POKE 832+I,X: REM SCHREIBEN DES BYTES IN DEN BLOCK
NEXT I: REM ENDE DER SCHLEIFE
```



Danach können alle Informationen über das Sprite mit einem oder zwei POKES verändert werden.

## POSITION

Als erstes wollen wir uns nun mit der Lage des Sprites auf dem Bildschirm befassen. Für die Position auf dem Bildschirm, hat jedes Sprite zwei Register. Das erste Register für Sprite 0 ist auch das Register 0. In diesem Register steht die Position des Sprites auf der X-Achse (die horizontale Achse). Im nächsten Register steht dementsprechend die Position auf der Y-Achse (vertikale Achse).

In unseren Beispielen, wird ab jetzt immer die Variable V (für Video Controller) verwendet. Diese Variable hat den Wert 53248. Diese Adresse ist die Anfangsadresse des Video Controllers. Ihre erste Zeile in jedem Programm, in dem Sie Graphik oder Sprites verwenden, sollte daher lauten:

```
V=53248: REM START DES VIDEO CONTROLLERS
```

Zur Positionierung des Sprites auf dem Bildschirm genügen nun zwei POKE Befehle:

```
POKE V+0, SPALTE: REM SPRITE 0 - X
POKE V+1, ZEILE: REM SPRITE 0 - Y
```

Wenn wir also unseren Hubschrauber in die Mitte des Bildschirms positionieren wollen, genügt:

```
POKE V+0,160: POKE V+1,120
```

## VERSCHIEBEN DER SPRITES

Von dieser Position aus, können wir das Sprite natürlich auch genauso leicht an einen anderen Punkte bewegen. Um eine fließende Bewegung zu erzeugen, müssen wir die jeweiligen Register um 1 verändern. Dies geschieht durch die Verwendung einer FOR NEXT Schleife. Zum Beispiel so:

```
FOR I=159 TO 100 STEP -1
  POKE V+0,I
NEXT I
```

Diese Routine verschiebt den Hubschrauber um ein ganzes Stück nach links. Er bleibt aber trotzdem auf der selben, horizontalen Achse. Wenn Sie die Bewegung des Sprites sichtbar machen wollen, müssen Sie vor dem NEXT I noch eine weitere FOR NEXT Schleife einsetzen:

```
FOR II=1 TO 100
  NEXT II
```

Nun verschiebt sich das Sprite wesentlich langsamer, und Sie können seine Bewegung beobachten.

Ein Problem werden Sie vielleicht schon erkannt haben: Es gibt 320 horizontale Positionen, aber der maximale Wert bei einem POKE ins X-Register kann nur 255 betragen. Wie kann man das Sprite nun an den rechten Bildschirmrand bringen ?

Für diese Aufgabe gibt es ein weiteres Register. Im Register 16 befindet sich für jedes Sprite ein besonderes Bit. In diesem Bit wird markiert, ob eine X-Koordinate verwendet werden soll, die über 255 liegt. Zu diesem Zweck, wird vor der Adressierung, das entsprechende Bit auf 1 gesetzt. Für unseren Hubschrauber hieße das:

POKE V+16,1

Nach diesem POKE wird zu unserer Adressierung jeweils der Wert 255 dazugaddiert. Nun würde ein POKE V+0,1 also eine Positionierung auf dem Punkt 256 bedeuten. Um wieder an Punkte kleiner 256 zu gelangen, muß dieses Bit wieder zurückgesetzt werden:

POKE V+16,0

Mit Hilfe dieser Einrichtung, ist es nun leicht möglich, jedes Sprite beliebig und unabhängig voneinander über den Bildschirm wandern zu lassen.

#### FARBE

Als weitere Möglichkeit, können wir noch die Farbe des Sprites ändern. Hierzu verfügt jedes Sprite über ein Farbregister. Diese Register sind die Register 39 bis 46:

Register:	39	40	41	42	43	44	45	46
Sprite:	0	1	2	3	4	5	6	7

Farben:

0	Schwarz
1	Weiss
2	Rot
3	Türkis
4	Violett
5	Grün
6	Blau
7	Gelb
8	Orange
9	Braun
10	Hell Rot
11	Grau 1
12	Grau 2
13	Hell Grün
14	Hell Blau
15	Grau 3

Unser Sprite bekommt also durch den Befehl:

POKE V+39,14

eine hellblaue Farbe.

#### VERGRÖßERUNG

Die nächste Besonderheit des Commodore 64, ist die Möglichkeit, die Sprites in horizontaler und / oder vertikaler Richtung zu vergrößern. Auch für diese Vergrößerung gibt es zwei Register. Eines für die Vergrößerung in X- und das andere für die

Vergrößerung in Y-Richtung. Es gibt aber nur diese beiden Register, die für alle Sprites verantwortlich sind. Daraus ergibt sich die Notwendigkeit, das jeweilige Sprite zu codieren.

Jedes Sprite-Kontroll-Register, ist daher folgendermaßen aufgebaut:

Byte:	7	6	5	4	3	2	1	0
Sprite:	7	6	5	4	3	2	1	0

Soll nun das erste Sprite (also Sprite 0), sowohl in X- als auch in Y-Richtung vergrößert werden, so sind folgende POKE Befehle notwendig:

```
POKE V+23,1: REM VERGRÖßERT SPRITE 0 IN Y-RICHTUNG
POKE V+29,1: REM VERGRÖßERT SPRITE 0 IN X-RICHTUNG
```

Um Sprite 0 und Sprite 1 in Y-Richtung zu vergrößern, brauchen Sie nur einen POKE V+23,3 anzuwenden usw.

Jedes Sprite kann in X- und / oder Y-Richtung um den Faktor 2 vergrößert werden. Das bedeutet, daß zum Beispiel bei der Erzeugung von Laufschrift, ein Zeichen insgesamt um den Faktor 4 vergrößert werden kann.

#### HINTERGRUND

Das nächste Beispiel demonstriert eine weitere Besonderheit. Sie können wählen, ob das Sprite vor oder hinter dem Hintergrund platziert wird. Dies kann natürlich hübsche Effekte hervorrufen. Wenn Sie unseren Hubschrauber auf dem Bildschirm haben, versuchen Sie einmal den Befehl:

```
POKE V+27,1
```

Durch diesen Befehl, sagen Sie dem Sprite, ob es vor oder hinter dem Hintergrund erscheinen soll. Um eine Reaktion zu erkennen, müssen Sie mit dem Cursor in die Zeile gehen, in der sich das Sprite auf dem Bildschirm befindet. Dann geben Sie einfach ein paar Zeichen ein, gerade soviel, daß das Sprite von den Zeichen überdeckt wird. Probieren Sie doch einfach einmal aus, welche Zeichenfarbe den besten Kontrast zu dem Sprite bildet. Sie sehen nun, daß die Schrift über dem Sprite liegt. Tatsächlich ist es so, daß Sie das Sprite in eine andere Ebene gebracht haben, nämlich unter den Hintergrund. Gehen Sie in eine andere freie Zeile und schreiben Sie:

```
POKE V+27,0
```

Nun schiebt sich das Sprite wieder vor den Text. Das Register für die Hintergrund-Sprite Priorität, ist also so aufgebaut:

Byte:	b7	b6	b5	b4	b3	b2	b1	b0
Prior.:	7	6	5	4	3	2	1	0

Mit diesem Register, können Sie so die Priorität jedes Sprites beliebig verändern. Wenn das entsprechende Bit nicht gesetzt, also 0 ist, bedeutet das, daß das Sprite vor dem Hintergrund steht. Ein gesetztes Bit bedeutet demnach, daß das Sprite hinter dem Hintergrund steht. Durch unterschiedliches setzen und löschen der Bits, besteht die Möglichkeit, mehrere Sprites übereinander

zu legen, und trotzdem deutlich zwischen Vorder- und Hintergrund unterscheiden zu können.

Wie wir eben schon erwähnten, läßt sich durch diese Möglichkeit des Commodore 64, leicht eine dreidimensionale Darstellung erreichen.

#### KOLLISION: SPRITE - SPRITE

Zu dem übereinanderschieben von Sprites gibt es aber noch etwas anderes zu sagen. Es gibt ein Register, in dem eine mögliche Kollision zwischen verschiedenen Sprites verzeichnet wird. Dieses Register bleibt solange auf 0, bis entweder zwei oder mehrere Sprites zusammengestoßen sind, oder Sie dieses Register mit einem POKE Befehl zurücksetzen. Denn haben sich erst einam Sprites überdeckt, fand also eine Kollision statt, bleibt dieses Register mit den Informationen über die zusammengestoßenen solange geladen, bis Sie es zurücksetzen. Der Wert dieses Registers gibt Auskunft darüber, welche Sprites kollidiert sind. Würde bei einer Abfrage

PRINT PEEK(V+30) oder KO=PEEK(V+30)

zum Beispiel der Wert 3 erscheinen, so wissen Sie dann, daß die Sprites 1 und 2 zusammengestoßen sind. Die Bitstruktur des Registers ist im übrigen genauso, wie bei allen anderen Registern auch. Sie braucht daher hier nicht erläutert zu werden. Nach dieser Abfrage müssen Sie dann durch

POKE V+30,0

das Register wieder zurücksetzen. Ansonsten würden diese beiden Sprites weiterhin als kollidiert erkannt werden, selbst wenn dieses nicht der Fall sein sollte.

#### KOLLISION: SPRITE - HINTERGRUND

Genauso, wie eine Kollision von verschiedenen Sprites registriert wird, besteht die Möglichkeit auf eine Kollision von Sprites mit dem Hintergrund zu reagieren. Dazu dient das Register 31. Dieses Register wird gleich dem Register 30 behandelt. Die einzige Ausnahme ist das Ergebnis der PEEK Abfrage. Das Resultat gibt nur Auskunft darüber, welche(r) Sprite(s) mit einem Zeichen kollidiert sind. Nicht aber welches Zeichen es war, und an welcher Position es sich befand. Um diese Informationen zu bekommen, müssen Sie die entsprechenden anderen Register und Speicherstellen abfragen. Zur Verdeutlichung noch einmal die Syntax (Sprache) des Befehls:

PRINT PEEK(V+31) oder KO=PEEK(V+31)

Es braucht wohl nicht darauf hingewiesen zu werden, daß auch dieses Register nach einer Kollision wieder zurückgesetzt werden muß.

#### VERSCHIEBEN DES BILDSCHIRMS

Zwei weitere interessante Register sind die Register 17 und 22. Durch Ihre Verwendung, läßt sich der gesamte Bildschirm Schritt für Schritt verschieben. Diese Verschiebung beschränkt sich auf 8 Schritte nach oben, unten, rechts, und links. Register 17 ist für

die Verschiebung in Y-Richtung zuständig, während Register 22 eine Verschiebung in X-Richtung steuert. Bei der Benutzung dieser Einrichtung ist auf 2 Dinge zu achten:

1.) Das Bit 3 des entsprechenden Registers muß gesetzt sein - erst dann kann eine ordnungsgemäße Verschiebung erfolgen.

2.) Bei einer Veränderung der Registerinhalte, darf nicht einfach ein neuer Wert in diese Register geschrieben werden, da sonst auch andere Bits verändert werden können. Der jeweilige POKE muß mit dem alten Registerinhalt odert werden.

Beispiel:

Setzen von Bit 3 im Register 22:

POKE V+22,PEEK(V+22) OR 8

Mit dieser Anwendung, läßt sich dann der gesamte Bildschirminhalt um den angegebenen Wert verschieben.

#### MULTICOLOR

Der Clou bei der Programmierung der Sprites, ist die Möglichkeit der Definition eines Sprites als Multicolor. Das Sprite kann dann aus maximal 3 Farben bestehen, hat aber eine geringere Auflösung, da jeweils zwei Bit als ein Punkt betrachtet werden. Daraus ergibt sich dann anstatt einer 8 x 8 eine 4 x 8 Matrix. In diesen zwei Bits befinden sich dann die Informationen über die Farben und dadurch auch die Information, ob dieser Punkt gesetzt ist oder nicht.

Wir wissen ja nun, daß man mit 2 Bit insgesamt 4 Informationen übermitteln kann: 00, 01, 10 und 11. Bei einer Verwendung von Multicolor, haben diese 2 Bits folgende Wirkung:

- 00 - Der Punkt hat die Farbe des Hintergrundes  
(man sieht also keinen Punkt)
- 01 - Die Farbe wird aus dem Register 37 geholt  
(der Punkt hat dann die entsprechende Farbe)
- 10 - Die Farbe wird aus dem Sprite-Farbregister geholt  
(die Farbe aus dem entsprechenden Register 39-46)
- 11 - Die Farbe wird aus dem Register 38 geholt  
(der Punkt hat dann die entsprechende Farbe)

Sie verstehen nun, daß das Sprite aus einer eigenen und zwei Farben, die alle Sprites gemeinsam haben, bestehen kann. Sieht man einmal davon ab, daß ein nicht gesetzter Punkt ja auch eine Farbe hat, nämlich die Hintergrundfarbe.

Vielleicht haben Sie sich schon gefragt, warum unser Hubschrauber etwas seltsam aussieht. Diese Frage können wir nun beantworten. Er ist als Multicolor-Sprite entwickelt worden. Da ein Multicolor-Sprite aus weniger Punkten besteht, als ein einfarbiges Sprite, kann man natürlich im normalen Sprite-Modus kein vernünftiges Bild bekommen. Als Abschluß dieses Kapitels, wollen wir Ihnen nun das fertige Programm, mit der Darstellung des Multicolor-Hubschraubers präsentieren. Es ist nicht ganz ungeschickt, an Hand dieses Programmes, mit der Programmierung

der Sprites zu experimentieren. Dadurch erlernen Sie die Handhabung der Sprites am schnellsten.

```
10 REM SPRITE DEMONSTRATION - HUBSCHRAUBER
20 V=53248: REM ANFANG VIDEO CONTROLLER
30 POKE V+32,15: POKE V+33,14: REM HINTERGRUNDFARBEN
40 PRINT "<CTRL>-7": REM DRÜCKEN SIE CONTROL UND 7 GLEICHZEITIG
50 POKE V+21,3: REM ERÖFFNEN VON SPRITE 0 UND 1
60 POKE V+28,3: REM SPRITE 0 UND 1 SIND MULTICOLOR
70 POKE V+39,6: REM FARBE VON SPRITE 0 - BLAU
80 POKE V+40,2: REM FARBE VON SPRITE 1 - ROT
90 POKE V+37,14: REM MULTICOLOR-FARBE 1 - HELL BLAU
100 POKE V+38,0: REM MULTICOLOR-FARBE 2 - SCHWARZ
110 POKE 2040,13: REM SPRITE 0 AUS BEREICH 832 BIS 895
120 POKE 2041,13: REM SPRITE 1 AUS BEREICH 832 BIS 895
130 FOR I=0 TO 62: REM SCHLEIFE ZUM EINLESEN DER DATEN
140 : READ X: REM LESEN DER PUNKTEKOMBINATIONEN
150 : POKE 832+I,X: REM SPEICHERN DER PUNKTEKOMBINATION
160 NEXT I: REM ENDE DER SCHLEIFE
170 POKE V+0,24: POKE V+1,50: REM POSITION VON SPRITE 0
180 POKE V+2,60: POKE V+3,50: REM POSITION VON SPRITE 1
190 END
1000 DATA 000,000,000
1010 DATA 000,000,000
1020 DATA 000,000,000
1030 DATA 000,000,000
1040 DATA 000,000,000
1050 DATA 000,000,000
1060 DATA 000,000,000
1070 DATA 003,255,255
1080 DATA 000,002,000
1090 DATA 192,170,128
1100 DATA 194,150,080
1110 DATA 234,150,080
1120 DATA 194,170,168
1130 DATA 192,170,168
1140 DATA 000,032,128
1150 DATA 000,170,160
1160 DATA 000,000,000
1170 DATA 000,000,000
1180 DATA 000,000,000
1190 DATA 000,000,000
1200 DATA 000,000,000
```

Dieses Programm können Sie nun wieder als Anregung zu eigenen Entwicklungen benutzen. Aber denken Sie immer an eines: Jedes Programm ist nur so gut, wie die Vorbereitung dazu war. Unterschätzen Sie diesen Faktor nicht.

Um Ihnen bei der Erstellung von Sprites behilflich zu sein, haben wir ein Sprite Entwurfsblatt entwickelt. Das Entwurfsblatt finden Sie im ANHANG am Ende dieses Buches.

Dieses Entwurfsblatt können Sie entweder aus dem Buch kopieren, oder aber auch nur als Anregung für ein eigenes Entwurfsblatt verwenden. Sie werden sehen, daß Sie später überhaupt nicht mehr ohne ein solches Entwurfsblatt arbeiten wollen. Denn auch hier gilt die Regel: überlegen Sie sich vorher, was Sie programmieren wollen.

## ENTWURF

Wie benutzt man dieses Entwurfsblatt ? Als erstes machen Sie auf einem normalen Blatt Papier Ihre groben Entwürfe für das Sprite. Sie sollten dabei bedenken, ob das Sprite aus einer Farbe bestehen soll, oder ob es sich um ein Multicolor Sprite handelt. Als nächstes machen Sie auf Ihrem Entwurfsblatt den endgültigen Entwurf (abhängig von der Art des Sprites). Sie füllen also die Punkte entweder in der  $8 \times 8$  Matrix völlig aus, oder aber verwenden die 3 Farben, die Ihnen pro Seite zur Verfügung stehen.

## MULTICOLOR SPRITE

Wenn Sie sich für ein Multicolor Sprite entscheiden, denken Sie daran, daß zwei Punkte auf dem Blatt einen Punkt auf dem Bildschirm darstellen. Gleichzeitig müssen Sie sich für das jeweilige Register entscheiden, welches Sie verwenden wollen.

Es ist am günstigsten, wenn Sie bei diesem Entwurf, eine andere Form als bei den einfarbigen Sprites wählen (darauf kommen wir gleich noch). Das heißt, daß Sie nicht jeden Punkt voll ausfüllen, sondern in die zwei Punkte die Bitkombination eintragen, die für die Register zuständig ist. Wie diese Bitkombination aussieht, haben wie Ihnen im Abschnitt über die Multicolor Sprites bereits gezeigt. Wenn Sie das ganze Arbeitsblatt ausgefüllt haben, geht es ans Abzählen der Bits. Wir haben Ihnen durch die Angabe der Dezimalzahlen, diese Abzählung schon erleichtert. Wenn Sie also zählen, brauchen Sie nur die Zahlen zu addieren, deren Bit auf 1 gesetzt ist. So erhalten Sie in jeder Zeile 3 dezimale Zahlen, die Sie wie in unserem Beispielsprogramm nebeneinander aufschreiben (die DATA Zeilen). Dieses führen Sie insgesamt 21 mal aus, und erhalten so Ihr gesamtes Sprite in den DATA Zeilen.

## EINFARBIGE SPRITES

Bei den einfarbigen Sprites, ist das alles etwas einfacher. Hier können Sie jeden Punkt auf dem Blatt als einen Punkt auf dem Bildschirm behandeln. So ist es hier möglich, jeden Punkt mit einem Stift vollständig auszufüllen, und so einen klaren Eindruck über das endgültige Sprite zu erhalten. Danach übertragen Sie die Zahlen wieder in die DATA Zeilen. Ihr Sprite ist damit programmiert, und fertig für den Einsatz.

## Kapitel 4 : EIN- AUSGABE STEUERUNG - CIA 6526

### 4.1 Allgemeines über den 6526

Der Complex Interface Adapter (CIA) 6526 ist ein neuer Peripheriebaustein aus der 65xx-Familie. Er verfügt über:

- \* 16 einzeln programmierbare Ein- Ausgabelleitungen
- \* 8 oder 16 Bit Handshake sowohl bei Eingabe als auch bei Ausgabe
- \* 2 unabhängige, kaskadierbare 16 Bit Intervalltimer
- \* 24 Stunden (AM/PM) Uhr mit programmierbarer Alarmzeit
- \* 8 Bit Schieberegister für die serielle Ein- Ausgabe

Das Blockschema der CIA 6526 finden Sie auf der nächsten Seite. Zur speziellen Belegung der CIAs im CBM64 sehen Sie bitte unbedingt im Kapitel 4.6 nach!

Pinbelegung des 40-poligen Gehäuses:

- 1 Masse
- 2 - 9 I/O-Port A; 8 Bit bidirektional
- 10-17 I/O-Port B; 8 Bit bidirektional. Die Bits 6+7 können zur Anzeige des Unterlaufs der beiden Timer programmiert werden.
- 18 -PC(Port Control); nur Ausgang; signalisiert die Verfügbarkeit von Daten am Port B oder an beiden Ports.
- 19 TOD(Time Of Day); nur Eingang 50/60 Hz; triggert die Echtzeituhr.
- 20 +5V; Betriebsspannung
- 21 -IRQ(Interrupt Request); nur Ausgang; wird 0 bei Übereinstimmung eines gesetzten Bits im ICR mit dem Eintreffen des zugehörigen Ereignisses.
- 22 R/W(Read/-Write); nur Eingang; 0=Übernahme des Datenbus, 1=Ausgabe auf den Datenbus.
- 23 -CS(Chip Select); nur Eingang; 0=Datenbus gültig, 1=Datenbus hochohmig(Tri-State).
- 24 -FLAG; nur Eingang; Bedeutung wie -PC.
- 25 02(Systemtakt 2); nur Eingang; alle Datenbusaktionen finden nur bei 02=1 statt.
- 26-33 DB7-DB0(Datenbus); bidirektional; Schnittstelle zum Prozessor.
- 34 -RES(Reset); nur Eingang; 0=Rücksetzen der CIA in den Grundzustand.
- 35-38 RS3-RS0(Register Select); nur Eingang; dient zur Auswahl eines der 16 Register der CIA; nur gültig mit -CS=0.
- 39 SP(Serial Port); bidirektional; dient als Ein- Ausgang des Schieberegisters
- 40 CNT(Count); bidirektional; Ein- Ausgang des Schieberegistertakts oder Triggereingang für die Intervalltimer.

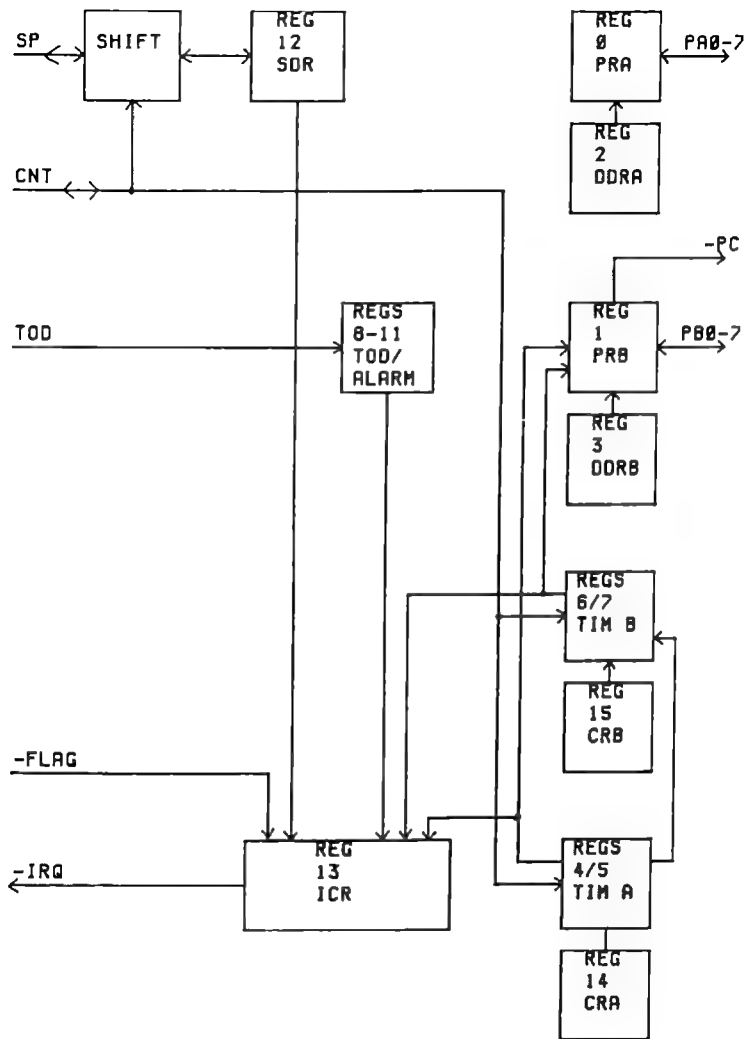
### 4.2 Registerbeschreibung der CIA

REG 0 PRA (Port Register A)

Zugriff: READ/WRITE

Bit 0-7 Dieses Register entspricht dem Zustand der Pins PA0-7





- REG 1 PRB (Port Register B)  
 Zugriff: READ/WRITE  
 Bit 0-7 Dieses Register entspricht dem Zustand der Pins PB0-7
- REG 2 DDRA (Datenrichtung Register A)  
 Zugriff: READ/WRITE  
 Bit 0-7 Diese Bits bestimmen die Datenrichtung der korrespondierenden Datenbits des Ports A. 0=Eingang, 1=Ausgang.
- REG 3 DDRB (Datenrichtung Register B)  
 Zugriff: READ/WRITE  
 Bit 0-7 Diese Bits bestimmen die Datenrichtung der entsprechenden Datenbits des Ports B. 0=Eingang, 1=Ausgang.
- REG 4 TAIL (Timer A LO-Byte)  
 Zugriff: READ  
 Bit 0-7 Dieses Register gibt den augenblicklichen Zustand des niederwertigen Byte von Timer A wieder.  
 Zugriff: WRITE  
 Bit 0-7 In dieses Register wird das niederwertige Byte des Wertes geladen, von dem der Timer auf null zählen soll.
- REG 5 TAIH (Timer A HI-Byte)  
 Zugriff: READ  
 Bit 0-7 Dieses Register gibt den augenblicklichen Zustand des höherwertigen Byte von Timer A wieder.  
 Zugriff: WRITE  
 Bit 0-7 In dieses Register wird das höherwertige Byte des Wertes geladen, von dem der Timer auf null zählen soll.
- REG 6 TBL (Timer B LO-Byte)  
 Zugriff und Belegung entspricht REG 4.
- REG 7 TBH (Timer B HI-Byte)  
 Zugriff und Belegung entspricht REG 5.
- REG 8 TOD10THS (Uhr 1/10 sec)  
 Zugriff: READ  
 Bit 0-3 Zehntelsekunden der Echtzeituhr im BCD-Format.  
 Bit 4-7 Immer 0.  
 Zugriff: WRITE und CRB Bit 7=0  
 Bit 0-3 Zehntelsekunden im BCD-Format.  
 Bit 4-7 Müssen 0 sein.  
 Zugriff: WRITE und CRB Bit 7=1  
 Bit 0-3 Vorwahl der Zehntelsekunden der Alarmzeit im BCD-Format.  
 Bit 4-7 Müssen 0 sein.

- REG 9 TOD SEC (Uhr sec)  
 Zugriff: READ  
 Bit 0-3 Einersekunden der Uhr im BCD-Format.  
 Bit 4-6 Zehnersekunden der Uhr im BCD-Format.  
 Bit 7 Immer 0.  
 Weitere Zugriffsarten analog zu REG 8.
- REG 10 TOD MIN (Uhr min)  
 Zugriff: READ  
 Bit 0-3 Einerminuten der Uhr im BCD-Format.  
 Bit 4-6 Zehnerminuten der Uhr im BCD-Format.  
 Bit 7 Immer 0.  
 Weitere Zugriffsarten analog zu REG 8.
- REG 11 TOD HR (Uhr Std)  
 Zugriff: READ  
 Bit 0-3 Einerstunden der Uhr im BCD-Format.  
 Bit 4 Zehnerstunde der Uhr.  
 Bit 5-6 Immer 0.  
 Bit 7 0=vormittags(AM), 1=nachmittags(PM).  
 Weitere Zugriffsarten analog zu REG 8.
- REG 12 SDR (Serial Data Register)  
 Zugriff: READ/WRITE  
 Bit 0-7 Aus diesem Register werden die Daten bitweise zum Pin SP hinausgeschoben, bzw. vom Pin 9P in dieses Register hineingeschoben.
- REG 13 ICR (Interrupt Control Register)  
 Zugriff: READ (INT DATA)  
 Bit 0 1=Unterlauf Timer A.  
 Bit 1 1=Unterlauf Timer B.  
 Bit 2 1=Gleichheit von Uhrzeit und gewählter Alarmzeit.  
 Bit 3 1=SDR voll/leer (abhängig von der Betriebsart).  
 Bit 4 1=Signal am Pin FLAG aufgetreten.  
 Bit 5-6 Immer 0.  
 Bit 7 Übereinstimmung mindestens eines Bits von INT MASK und INT DATA aufgetreten.  
 ACHTUNG: Beim Lesen dieses Registers werden alle Bits gelöscht.  
 Zugriff: WRITE (INT MASK)  
 Bedeutung der Bits wie oben, ausgenommen Bit 7:  
 Bit 7 1=jedes 1-Bit setzt das korrespondierende Masken-Bit. Die anderen bleiben unberührt.  
 0=jedes 1-Bit löscht das korrespondierende Masken-Bit. Die anderen bleiben unberührt.
- REG 14 CRA (Control Register A)  
 Zugriff: READ/WRITE  
 Bit 0 1=Timer A Start, 0=Stop  
 Bit 1 1=Unterlauf von Timer A wird an Pin PB6 signalisiert.  
 Bit 2 1=jeder Unterlauf von Timer A kippt PB6 in die jeweils andere Lage, 0=jeder Unterlauf von Timer A erzeugt an PB6 einen HI-Puls mit der Länge eines Systemtaktes.

Bit 3 1=Timer A zählt nur einmal vom Ausgangswert auf null und hält dann an, 0=Timer A zählt fortlaufend vom Ausgangswert auf null.

Bit 4 1=unbedingtes Laden eines neuen Startwertes in Timer A. Dieses Bit fungiert als Strobe. Es muß bei jedem unbedingten Laden neu gesetzt werden.

Bit 5 Dieses Bit bestimmt die Quelle des Timer-Triggers. 1=Timer zählt steigende CNT-Flanken, 0=Timer zählt Systemtakte.

Bit 6 1=SP ist Eingang, 0=SP ist Ausgang.

Bit 7 1=Echtzeituhr-Trigger beträgt 50Hz, 0=Trigger beträgt 60 Hz.

#### REG 15 CRB (Control Register B)

Zugriff: READ/WRITE

Bit 0-4 Diese Bits haben die gleiche Bedeutung wie in REG14, allerdings bezogen auf Timer B und Pin PB7.

Bit 5-6 Diese Bits bestimmen die Quelle des Triggers für Timer B. 00=Timer zählt Systemtakte, 10=Timer zählt steigende CNT-Flanken, 01=Timer B zählt Unterläufe von Timer A, 11=Timer B zählt Unterläufe von Timer A, wenn CNT=1 ist.

Bit 7 1=Alarm setzen, 0=Uhrzeit setzen.

### 4.3 E/A-Ports

Die Ports A und B bestehen je aus einem 8-Bit Datenregister (PR) und einem 8-Bit Datenrichtungsregister (DDR). Wenn ein Bit im DDR gesetzt ist, arbeitet das korrespondierende Bit im PR als Ausgang. Ist ein Bit im DDR =0, ist das entsprechende Bit im PR als Eingang definiert.

Während eines Lesezugriffs gibt das PR den augenblicklichen Zustand der entsprechenden Pins (PA0-7, PB0-7) wieder, und zwar sowohl für die Eingangs- als auch für die Ausgangsbits. Darüberhinaus können PB6 und PB7 noch Ausgangsfunktionen für die beiden Timer übernehmen.

Der Datentransfer zwischen der CIA und der an PA/PB angeschlossenen "Außenwelt" kann durch einen Quittungsbetrieb erreicht werden. Hierzu dienen PC und FLAG.

PC wird für die Dauer eines Taktes 0, wenn ein Lese- oder Schreibzugriff auf PRB vorangegangen ist. Dieses Signal kann so die Verfügbarkeit von Daten an PB, bzw. die Annahme von Daten von PB anzeigen.

FLAG ist ein negativ flankengetriggert Eingang, der z.B. mit PC einer anderen CIA verbunden werden könnte. Eine fallende Flanke an FLAG setzt auch das FLAG-Interrupt-Bit.

Der serielle Datenport SDR ist ein synchrones 8-Bit Schieberegister. CRA Bit6 bestimmt Ein- oder Ausgabemodus.

Im Eingabemodus werden die Daten an SP mit der steigenden Flanke eines an CNT liegenden Signales in ein Schieberegister übernommen. Nach 8 CNT-Pulsen wird der Inhalt des Schieberegisters nach SDR gebracht und das SP-Bit im ICR gesetzt.

Im Ausgabemodus fungiert Timer A als Baudrate-Generator. Die Daten aus SDR werden mit der halben Unterlauffrequenz von

Timer A nach SP hinausgeschoben. Die theoretisch höchste Baudrate beträgt demnach 1/4 des Systemtaktes. Die Übertragung beginnt, nachdem Daten ins SDR geschrieben wurden, vorausgesetzt Timer A läuft und befindet sich im Continuous-Modus (CRA Bit 0=1 und Bit 3=0). Der von Timer A abgeleitete Takt erscheint an CNT. Die Daten aus SDR werden in das Schieberegister geladen und dann mit jeder fallenden Flanke an CNT aus SP hinausgeschoben. Nach 8 CNT-Pulsen wird der SP-Interrupt erzeugt. Wird jedoch SDR vor diesem Ereignis mit neuen Daten geladen, so werden diese nun automatisch ins Schieberegister geladen und hinausgeschoben. In diesem Falle erscheint kein Interrupt. Die Daten aus SDR werden mit dem höchstwertigen Bit voran hinausgeschoben. Eingehende Daten sollten dasselbe Format aufweisen.

#### 4.4 Timer

Jeder der beiden Intervalltimer besteht aus einem 16-Bit Zähler (read only) und einem 16-Bit Zwischenspeicher (write only).

Daten, die in den Timer geschrieben werden, landen im Zwischenspeicher, während die Lesedaten den augenblicklichen Stand des Zählers wiedergeben.

Die beiden Timer können sowohl unabhängig voneinander als auch im Zusammenhang benutzt werden. Die verschiedenen Betriebsarten erlauben das Erzeugen langer Zeitverzögerungen, variable Pulslängen und Impulsketten. Bei Benutzung des CNT-Eingangs können die Timer externe Impulse zählen oder Frequenzen messen.

Jeder Timer hat ein ihm zugeordnetes Steuerregister (CRA/CRB), welches die folgenden Funktionen erlaubt:

##### START/STOP (Bit 0)

Dieses Bit läßt den Timer jederzeit starten oder anhalten.

##### PB ON/OFF (Bit 1)

Hiermit wird der Timer-Unterlauf nach PB geleitet (PB6 für Timer A, PB7 für Timer B). Diese Funktion hat Vorrang vor der in DDRB festgelegten Datenrichtung.

##### TOGGLE/PULSE

Mit diesem Bit wird die Art des an PB erscheinenden Unterlaufpulses bestimmt. Entweder wird PB bei jedem Unterlauf in die jeweils andere Lage gekippt, oder es wird ein positiver Puls mit der Dauer eines Taktes erzeugt.

##### ONE-SHOT/CONTINUOUS (Bit 3)

Im One-Shot-Betrieb zählt der Timer vom Zwischenspeicherwert nach null, setzt das IRC-Bit, lädt den Zähler erneut mit dem Zwischenspeicherwert und hält dann an. Im Continuous-Modus läuft der oben beschriebene Vorgang zyklisch ab.

##### FORCE LOAD (Bit 4)

Dieses Bit erlaubt den Timer jederzeit zu laden, gleichgültig ob er gerade läuft oder nicht.

#### INPUT MODE (Bit 5 CRA, Bit 5-6 CRB)

Diese Bits erlauben die Wahl des Taktes, mit dem der Timer heruntergezählt wird. Timer A kann entweder mit dem Systemtakt oder mit einem auf CNT gegebenen Takt versorgt werden. Timer B kann darüberhinaus noch mit den Unterlauf-Pulsen von Timer A gespeist werden, entweder unbedingt oder in Abhängigkeit von CNT=1.

### 4.5 Echtzeituhr

Die Echtzeituhr (TOD) ist eine 24-Stunden-Uhr (AM/PM) mit einer Auflösung von 1/10sec.

Sie besteht aus vier Registern: 1/10sec, sec, min, Std. Das AM/PM-Bit ist das höchstwertige des Stundenregisters. Jedes Register ist im BCD-Format organisiert, sodaß die gelesenen Werte ohne große Rechenoperationen benutzt werden können.

Als Takt dient ein 50/60 Hz-Signal (programmierbar, CRA Bit 7) am Pin TOD.

Außerdem gibt es noch ein Alarm-Register, mit welchem man zu jeder gewünschten Zeit einen Interrupt erzeugen kann. Das Alarmregister belegt dieselbe Adresse wie das TOD-Register. Deshalb wird der Zugriff mit CRB Bit 7 gesteuert. Das Alarmregister ist write only. Jeder Lesezugriff gibt den Stand des TOD-Registers wieder, unabhängig von CRB Bit 7.

Um die Uhrzeit korrekt setzen und lesen zu können, muß eine bestimmte Reihenfolge eingehalten werden:

Wenn das Stundenregister beschrieben wird, hält die Uhr automatisch an. Erst wenn ein Schreibzugriff auf das 1/10sec-Register erfolgt ist, läuft die Uhr weiter. Hierdurch startet die Uhr tatsächlich zur gewünschten Zeit. Da während des Lesens der vollständigen Uhrzeit ein Übertrag in ein bereits gelesenes Register auftreten kann, wird beim Lesen des Stundenregisters die gesamte Uhrzeit in einem Zwischenspeicher gepuffert. Der Zwischenspeicher wird erst wieder freigegeben, wenn die 1/10sec gelesen wurden.

Soll nur ein Register gelesen werden, so kann dies selbstverständlich auch 'en passant' geschehen; sollte es sich bei diesem Register jedoch um das Stundenregister handeln, so muß anschließend das 1/10sec-Register gelesen werden, um den Zwischenspeicher wieder freizugeben.

#### 4.5.1 Mit einem Trick die richtige Zeit

Die Langzeitgenauigkeit der vom Betriebssystem versorgten Uhr T1s läßt systembedingt zu wünschen übrig.

Es ist mit einem maximalen Fehler von 1/28td. pro Tag zu rechnen.

Für diejenigen, die auf eine genaue Uhrzeit Wert legen, bietet sich die in den CIAs enthaltene Echtzeituhr an.

Diese erhält ihren Takt aus der Netzfrequenz, welche eine hervorragende Langzeitkonstanz aufweist.

Um Ihnen die Handhabung der Echtzeituhr zu erleichtern, haben wir zwei kleine Basicprogramme ausgearbeitet.

Das eine dient zum Stellen der Uhr, das andere zum Auslesen. Hier nun zunächst das Programm zum Stellen. Der Wert für 1/10sec wird hierbei immer auf 0 gesetzt.

```

10 C=56328: REM Basisadr. der Uhr in CIA 1
20 REM C=56584 für Uhr in CIA 2
30 POKE C+7,PEEK(C+7)AND127
35 POKE C+6,PEEK(C+6)OR128
40 INPUT"ZEIT IM FORMAT HHMMSS EINGEBEN":A$
50 IF LEN(A$)<>6 THEN 40
60 H=VAL(LEFT$(A$,2))
70 M=VAL(MID$(A$,3,2))
80 S=VAL(RIGHT$(A$,2))
90 IF H>23 THEN 40
100 IF H>11 THEN H=H+60
110 POKE C+3,16*INT(H/10)+H-INT(H/10)*10
120 IF M>59 THEN 40
130 POKE C+2,16*INT(M/10)+M-INT(M/10)*10
140 IF S>59 THEN 40
150 POKE C+1,16*INT(S/10)+S-INT(S/10)*10
160 POKE C,0

```

Das Lesen der Uhrzeit ermöglicht folgendes Programm:

```

10 C=56328: REM Basisadr. der Uhr in CIA 1
20 PRINT "(shft/clr)":REM C=56584 für Uhr in CIA 2
30 H=PEEK(C+3):M=PEEK(C+2):S=PEEK(C+1):T=PEEK(C)
40 FL=1
50 IF H>32 THEN H=H-128:FL=0
60 H=INT(H/16)*10+H-INT(H/16)*16:ON FL GOTO 80
65 IF H=12 THEN 85
70 H=H+12
80 IF H=12 THEN H=0
85 M=INT(M/16)*10+M-INT(M/16)*16
90 S=INT(S/16)*10+S-INT(S/16)*16
100 T$=STR$(T)
110 H$=STR$(H):IF LEN(H$)=2 THEN H$=" 0"+RIGHT$(H$,1)
120 M$=STR$(M):IF LEN(M$)=2 THEN M$=" 0"+RIGHT$(M$,1)
130 S$=STR$(S):IF LEN(S$)=2 THEN S$=" 0"+RIGHT$(S$,1)
140 PRINT "(home)":
150 PRINT RIGHT$(H$,2):" ":RIGHT$(M$,2):" ":RIGHT$(S$,2):"0":
160 PRINT RIGHT$(T$,1)
170 GOTO 30

```

Nach Drücken von STOP/RESTORE müssen Sie die Uhrzeit wieder neu setzen, da das Betriebssystem alle Register auf den Ausgangswert setzt. Davon ist leider auch das Bit für die 50/60Hz-Auswahl betroffen.

Ihre Uhr würde stark zurückbleiben.

#### 4.6 Die CIAs im CBM64

Wenn Sie die CIAs im CBM64 für eigene Zwecke nutzen wollen, beachten Sie bitte, daß ihnen im Rechner bestimmte Aufgaben zugeordnet sind. Insbesondere gilt das für die Verwendung von Interrupts, aufgrund derer das Betriebssystem bestimmte

Routinen durchläuft. Ändern Sie also möglichst nie die Maske im ICR.

Hier nun also die im CBM64 getroffene Zuordnung der CIA's:

CIA 1    Basisadresse \$DC00 (\$6320)

REG 0    (PRA)

Bit 0-7    Im normalen Betrieb wird hier die  
Reihenauswahl der Tastaturmatrix getroffen.  
Allerdings sind einige Bits mit dem Controlport 1  
außen am Rechner verbunden. Dieser dient zum Anschluß  
von Joysticks oder Paddles:

Bit 0-4    Joystick 0. Reihenfolge: oben, unten, links,  
rechts, Taste

Bit 6-7    Auswahl Paddle-Set A/B. Es darf nur eines  
der beiden Bits =1 sein.

REG 1    (PRB)

Bit 0-7    Im normalen Betrieb erfolgt hier die  
Spaltenrückmeldung der Tastaturmatrix, falls eine  
Taste gedrückt war.

Bit 0-4    Dieselbe Funktion wie REG 0, allerdings für  
Controlport 2 (Joystick 1).

REG 13    (ICR)

Bit 4    Eingabedaten vom Kassettenport

CIA 2    Basisadresse \$DD00 (\$6576)

REG 0    (PRA)

Bit 0-1    VA 14-15 (höchstwertige Adressbits des  
Videoram).

Bit 2    TXD (nur in Verbindung mit einer  
RS232-Cartridge, sonst frei).

Bit 3    ATN (Ausgang serieller Bus)

Bit 4    CLOCK (Ausgang ser. Bus)

Bit 5    DATA (Ausgang ser. Bus)

Bit 6    CLOCK (Eingang ser. Bus)

Bit 7    DATA (Eingang ser. Bus)

REG 1    (PRB)

Bit 0-7    Gewöhnlich frei. Bei Aufstecken einer  
RS232-Cartridge erhalten sie jedoch die folgende  
Bedeutung:

Bit 0    RXD (Receive Data)

Bit 1    RTS (Request To Send)

Bit 2    DTR (Data Terminal Ready)

Bit 3    RI (Ring Indicator)

Bit 4    DCD (Data Carrier Detect)

Bit 6    CTS (Clear To Send)

Bit 7    DSR (Data Set Ready)

REG 13    (ICR)

Bit 4    RXD (nur bei RS232-Betrieb, sonst frei).



#### 4.7 Die Verwendung von Joysticks

Zur Benutzung von Joysticks muß man wissen, daß sie dieselben CIA-Bits belegen, die auch zur Abfrage der Tastatur im normalen Betrieb benötigt werden.

Um dennoch mit den Joysticks arbeiten zu können, muß die Tastaturabfrage solange außer Gefecht gesetzt werden. Dies kann nur innerhalb eines Programmes erfolgen, welches die Tastatur nicht benutzt, also keinen INPUT oder dergleichen beinhaltet.

Das folgende kleine Programm zeigt die Joystick-Funktionen zuverlässig an:

```
10 poke56322,224
20 j=peek(56320)
30 if(jand1)=0then?"oben"
40 if(jand2)=0then?"unten"
50 if(jand4)=0then?"links"
60 if(jand8)=0then?"rechts"
70 if(jand16)=0then?"knopf"
80 goto20
```

Dieses Programm erwartet den Joystick an Control-Port 2. Wird der Joystick an Control-Port 1 betrieben, muß die Adresse in Zeile 20 auf 56321 geändert werden.

Aus obigem Programm rettet Sie nur STOP/RESTORE. Die letzte Zeile in einem "seriösen" Programm sollte sein:

```
100 poke56322,255
```

Hiermit wird die Tastatur wieder entriegelt. Sie können natürlich auch diese Zeile vor jede Tastatur-Aktion setzen, wenn Ihr Programm sowohl die Tastatur als auch die Joysticks benötigt. Allerdings muß dann vor jeder Joystick-Abfrage ein Befehl wie in Zeile 10 stehen.

## Kapitel 5 : BASIC EINMAL ANDERS GESEHEN

### 5.1 So arbeitet der BASIC-Interpreter

Der Commodore 64 bietet auf ROM einen komfortablen BASIC-Interpreter, der auf dem seinerzeit von MICROSOFT (MICROSOFT BASIC) für den Commodore PET 2001 entwickeltem BASIC-Interpreter aufbaut. Das BASIC trägt die Versionsnummer 2.0 und entspricht damit von den Funktionen her der 3000er Serie von Commodore. Zuerst soll einmal kurz auf die Arbeitsweise des Interpreters eingegangen werden.

Wenn Sie eine Programmzeile eingeben, so prüft der Interpreter, ob BASIC-Befehls Worte in der Zeile vorkommen. Erkennt er ein Befehls Wort, so wandelt er das Wort in ein Byte um, den Interpretercode, auch 'Token' genannt. Danach wird die Zeile aufgrund Ihrer Zeilennummer an der richtigen Stelle in das Programm eingefügt. Bei der Programmausführung erkennt er einen Befehl nun an diesem Interpretercode. In diesem Ein-Byte-Code ist das höchstwertige Bit (Bit 7) immer gesetzt (der Wert ist also größer als 127). Aufgrund dieses Codes ermittelt er dann aus einer Tabelle die Adresse der Routine, die den Befehl als Unterprogramm ausführt. Dann wird in einer Schleife der nächste Befehl geholt und ausgeführt. Die Adressen der Befehle stehen in Kapitel 7, die Routinen selbst sind im Kapitel 8 ausführlich beschrieben.

Wenn Sie nun eigene Funktionen und Routinen verwenden wollen, müssen Sie zuerst etwas über die Datenspeicherung des Interpreters wissen. Vom BASIC werden drei Datentypen unterschieden: Reelle Zahlen, ganze Zahlen und Strings. Reelle Zahlen liegen im Bereich von +/- 1E-39 bis +/- 1E38. Integerzahlen können nur ganzzahlige Werte von -32768 bis 32767 annehmen. Strings sind Zeichenketten mit einer Länge von 0 bis 255 Zeichen.

Wie werden Variablen nun gespeichert ?

Jeder Variableneintrag belegt sieben Byte; die ersten beiden Bytes sind der Namen (ASCII-Code), und die nächsten fünf Byte geben den Wert an. Bei reellen Zahlen ist das erste Byte der Exponent, und die nächsten 4 Byte stellen die Mantisse dar (halblogarithmische Darstellung). Bei Integerzahlen werden nur zwei Byte benutzt, und zwar high und low Byte der 16-Bit Binärdarstellung. Bei Strings steht im ersten Byte die Länge des Strings (0 bis 255), und die nächsten beiden Byte sind die Adresse des Strings.

Zur Unterscheidung der Variablentypen wird bei Integervariablen im ersten und zweiten Buchstaben des Namens das höchste Bit gesetzt. Bei Strings ist das höchste Bit des zweiten Zeichens gesetzt.

Die Variablentabelle im RAM beginnt unmittelbar nach dem BASIC-Programm. Ein Zeiger in der Zeropage gibt die Adresse an (\$2D/\$2E, 45/46).

Wenn Sie folgende Wertzuweisungen machen

```

A = 10.3
B% = -23
C$ = "Comodore 64"

```

und dann die Speicherinhalte mit dem Monitor ansehen, ergibt sich folgendes Bild:

```

002D 03 08      Variablentabelle beginnt bei $0803
0803 41 00 B4 24 CC CC CD  Variable A, Fließkommawert 10.3
080A C2 B0 FF E9 00 00 00  Variable B%, Binärwert -23
0810 43 B0 0C F4 9F 00 00  Variable C$, Länge 12,
                           Adresse $9FF4

```

## 5.2 Von der Eingabe bis zur Verarbeitung

Wie lassen sich nun Variablen oder Ausdrücke von BASIC aus an Maschinenprogramme übergeben ?

Dazu gibt es im BASIC-Interpreter eine komfortable Routine, die einen beliebigen Ausdruck holt und auswertet. Sämtliche Arithmetik geschieht mit reellen Zahlen. Kommen Integerzahlen vor, so werden diese erst ins Fließkommaformat umgewandelt.

Diese Routine mit Namen FRMEVL (FormelAuswertung) hat die Adresse \$AD9E und wertet sowohl numerische als auch Stringparameter aus. Zur Unterscheidung wird ein Typflag gesetzt (Adresse \$0D, 13 - \$00 bedeutet numerisch, \$FF bedeutet String). Numerische Daten werden im Fließkommaakku 1, kurz FAC genannt, abgelegt. Der FAC steht ab Adresse \$61 (97) in der Zeropage. Zur arithmetischen Verknüpfung, z.B. Addition, steht ein zweiter Fließkommaakku zur Verfügung, kurz ARG genannt (ab Adresse \$69 dezimal 105). Das Ergebnis nach dem Aufruf einer solchen Routine steht grundsätzlich im FAC. Auch wird bei Funktionsaufrufen das Argument in FAC übergeben und das Ergebnis dort wieder abgeholt. Wurde ein Stringausdruck ausgewertet, so steht in \$64/\$65 (100/101) ein Zeiger auf den sogenannten Stringdescriptor, der wiederum Länge und Adresse des Strings enthält. Der Aufruf der Routine \$B475 holt die Stringlänge in den Akku und die Adresse ins X- (low Byte) und Y-Register (high Byte).

Sehen wir uns jetzt einige nützliche Routinen des BASIC-Interpreters an, die arithmetische Aufgaben ausführen.

Adresse	Funktion	
\$B853	Minus	FAC = ARG - FAC
\$B86A	Plus	FAC = ARG + FAC
\$BA28	Multiplikation	FAC = ARG * FAC
\$BB12	Division	FAC = ARG / FAC
\$BF78	Potenzierung	FAC = ARG hoch FAC

Die Adressen der weiteren Routinen sowie der Belegung der Zero-Page durch den Interpreter und das Betriebssystem finden Sie in Kapitel 7.

### 5.3 Machen Sie MEHR aus Ihrem BASIC

#### 5.3.1 Wie erweitere ich BASIC ?

Eigene Maschinenroutinen lassen sich außer über die **USR-** und **SYS-**Funktion noch eleganter ins BASIC einbinden. Sehen wir uns dazu die Stelle im Interpreter an, die ein BASIC-Statement holt und ausführt. Hier ist der entsprechende Auszug aus dem ROM-Listing:

```
A7E1 6C 0B 03  JMP ($030B); zeigt normalerweise auf $A7E4
A7E4 20 73 00  JSR $0073  ; nächstes Zeichen aus
                        ; BASIC-Text holen
A7E7 20 A7 ED  JSR $A7ED  ; Statement ausführen
A7EA 4C AE A7  JMP $A7AE  ; zurück zur Interpreterschleife
```

An dieser Stelle können wir nun eingreifen. In Adresse \$030B/\$0309 steht ein Zeiger, den wir auf eine eigene Routine zur Kodeprüfung setzen können. Eine übliche Methode ist es, eigene Befehlserweiterungen durch ein vorangestelltes Sonderzeichen, z.B. ein Ausrufungszeichen, zu kennzeichnen, so könnte

```
100 'PRINT
```

eine eigene modifizierte Druckroutine aufrufen. Unsere Routine prüft dann auf das Ausrufungszeichen. Wird es gefunden, kann in die eigene Routine verzweigt werden, ansonsten wird die Routine des BASIC-Interpreters aufgerufen. Ein entsprechender Programmausschnitt könnte so aussehen:

```
DECODE JSR $0073  ; CHRGET, nächstes Zeichen
        CMP #'!'  ; mit Sonderzeichen vergleichen
        BEQ FOUND ; Verzweigung zur eigenen Routine
        JSR $0079  ; CHRGET, Flags wieder setzen
        JMP $A7E7  ; Befehl des Interpreters ausführen
FOUND   JSR $0073  ; CHRGET, nächstes Zeichen holen
        JSR COMMAND; eigene Befehle ausführen
        JMP $A7AE  ; zurück zur Interpreterschleife
```

Der Zeiger in \$030B/\$0309 muß beim Initialisieren der Befehlserweiterung auf die Adresse **DECODE** des obigen Beispielprogramms gesetzt werden. Will man mehrere Befehle implementieren, so kann man noch eine Routine zur Unterscheidung der Befehlsworte einbauen, die die verschiedenen Befehlserweiterungen selektiert.

#### 5.3.2 HARDCOPY - RENEW - PRINT USING

Im folgenden finden Sie einige Anregungen zur Verwirklichung eigener Routinen.

##### Beispiel 1 ~ Hardcopy-Funktion

Die Hardcopy-Funktion hat den Zweck, den Bildschirminhalt auf den Drucker (Geratenummer 4) zu kopieren und kann direkt mit **SYS 9\*4096** aufgerufen werden.

```

9000 45 24          LDA #4          : HARDCODED FUNKTION
9002 45 24          STA #4          : Geräterummer des Druckers
9004 45 75          LDA #105        :
9006 45 28          STA #5          : Logische Filenummer
9008 45 20          LDA #0          : Adresse low des Bildschirms
900A 40 24          LDR #4          : Adresse high des Bildschirms
900C 45 71          STA TEMP        : als Zeiger merken
900E 44 72          STY TEMP + 1
9010 45 67          STA #NAME      : kein Filename
9012 45 64          STA #4          : Sek-addr. null
9014 20 20 FF       JSR OPEN        : Drucker File öffnen
9016 45 28          LDR #5          : Logische Filenummer des Drucker
9018 20 28 FF       JSR #ROUT      : Drucker als Ausgabegerät
901A 40 18          LDR #15        : Anzahl der Bildschirmzeilen
901C 45 00          LDA #0          : neue Zeile
901E 20 20 FF       JSR #ROUT      : an Drucker
9020 20 21 FF       JSR #STOP      : Stoptaste abfragen
9022 40 2E          BEQ EXIT        : gedruckt, dann beenden
9024 40 00          LDR #0
9026 45 71          LDA #TEMP, : Zeichen von Bildschirm holen
9028 45 67          STA #STORE
902A 45 0F          AND #40F
902C 45 67          ASL #STORE
902E 45 67          BIT #STORE      : Bildschirmkennung
9030 40 00          BPL # + 4       : in ASCII-Code umwandeln
9032 40 00          JRA #180
9034 40 00          BVS # + 4
9036 40 40          JRA #140
9038 20 21 FF       JSR #ROUT      : und zum Drucker schicken
903A 40 00          INY
903C 40 40          CFI #40         : Zeile zu Ende
903E 40 00          BNE #LOOP2
9040 45 66          TRB
9042 45 16          CLC             : ja, Zeiger auf nächste
9044 45 71          ADD TEMP        : Zeile setzen
9046 45 71          STA TEMP
9048 40 00          BCC # + 4
904A 45 72          INC TEMP + 1
904C 40 00          BEQ #           : schon alle Zeilen ausgegeben
904E 40 00          BNE #LOOP
9050 45 00          LDA #15
9052 20 21 FF       JSR #ROUT      : neue Zeile
9054 20 20 FF EXIT   JSR #CLRCH    : Ausgabe wieder auf Bildschirm
9056 45 75          LDA #125
9058 40 20 FF       JMP #CLOSE     : Druckportal schließen und fertig

```

Es folgt ein Ladeprogramm in BASIC

```

100 POKE 56,9*16 : CLR : FOR I = 36864 TO 36957
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 4,133,186,169,126,133,184,169, 0,160, 4
130 DATA 133,113,132,114,133,183,133,185, 32,192,255,166
140 DATA 184, 32,201,255,162, 25,169, 13, 32,210,255, 32
150 DATA 225,255,240, 46,160, 0,177,113,133,103, 41, 63
160 DATA 6,103, 36,103, 16, 2, 9,128,112, 2, 9, 64
170 DATA 32,210,255,200,192, 40,208,230,152, 24,101,113
180 DATA 133,113,144, 2,230,114,202,208,205,169, 13, 32
190 DATA 210,255, 32,204,255,169,126, 76,195,255
200 IF S <> 12023 THEN PRINT "FEHLER IN DATAS !!" : END

```

210 PRINT "OK !"

## Beispiel 2 - RE-NEW

Das folgende Programm kann dann nützlich sein, wenn man versehentlich ein Programm mit NEW gelöscht hat. Das Programm findet das Ende des gelöschten Programms und setzt die BASIC-Zeiger wieder auf den alten Wert, sofern danach keine neuen Programmzeilen eingegeben oder Variablen benutzt wurden. Die Startadresse ist hier 12\*4096+15\*256 gleich 52992.

```

                                ; RE-NEW FUNKTION
                                ; HOLT GELOESCHTES PROGRAMM WIEDER ZURUECK
CF00 A5 2B      LDA PRGSTRT ; BASIC-Programmstart
CF02 A4 2C      LDY PRGSTRT + 1
CF04 B5 22      STA TEMP    ; als Zeiger speichern
CF06 B4 23      STY TEMP + 1
CF08 A0 03      LDY #3
CF0A CB        NULL      INY
CF0B B1 22      LDA (TEMP),Y ; sucht Ende der ersten Zeile
CF0D D0 FB      BNE NULL    ; (Nullbyte)
CF0F CB        INY
CF10 9B        TYA
CF11 1B        CLC
CF12 65 22      ADC TEMP    ; Offset addieren
CF14 A0 00      LDY #0
CF16 91 2B      STA (PRGSTRT),Y ; als Zeiger auf nächste
CF18 A5 23      LDA TEMP + 1
CF1A 69 00      ADC #0      ; Zeile speichern
CF1C CB        INY
CF1D 91 2B      STA (PRGSTRT),Y
CF1F 8B        DEY        ; enthält jetzt null
CF20 A2 03      TO      LDY #3
CF22 E6 22      TDREIO   INC TEMP
CF24 D0 02      BNE # + 4 ; Programmende gleich
CF26 E6 23      INC TEMP + 1; drei Nullbytes suchen
CF28 B1 22      LDA (TEMP),Y
CF2A D0 F4      BNE TO
CF2C CA        DEX
CF2D D0 F3      BNE TDREIO
CF2F A5 22      LDA TEMP
CF31 69 02      ADC #2
CF33 B5 2D      STA PRGEND
CF35 A5 23      LDA TEMP + 1 ; Zeiger auf Programmende setzen
CF37 69 00      ADC #0
CF39 B5 2E      STA PRGEND + 1
CF3B 4C 63 A6   JMP CLR    ; CLR und ready.

```

Hier wieder ein Ladeprogramm in BASIC. Dieses Programm muß natürlich zuerst geladen und gestartet werden, ehe man es auf (anschließend geladene oder geschriebene) versehentlich gelöschte BASIC-Programme anwenden kann.

```

100 FOR I = 52992 TO 53053
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 165, 43,164, 44,133, 34,132, 35,160, 3,200,177
130 DATA 34,208,251,200,152, 24,101, 34,160, 0,145, 43
140 DATA 165, 35,105, 0,200,145, 43,136,162, 3,230, 34
150 DATA 208, 2,230, 35,177, 34,208,244,202,208,243,165

```

```

160 DATA 34,105, 2,133, 45,165, 35,105, 0,133, 46, 76
170 DATA 99,166
180 IF S <> 7000 THEN PRINT "FEHLER IN DATAS !!" : END
190 PRINT "OK !"

```

### Beispiel 3 - PRINT USING

Eine sehr nützliche Routine, die dem Interpreter des Commodore 64 fehlt, ist die formatierte Ausgabe von Zahlen, oft mit 'PRINT USING' bezeichnet. Die Routine arbeitet nach folgendem Konzept (siehe Adresse \$C900 - \$C90C): Zuerst wird ein numerischer Ausdruck in den Fließkommaakku geholt und dann mit der Routine des BASIC-Interpreters in einen ASCII-String aus Ziffern umgewandelt, der ab Adresse \$100 abgelegt wird. Hier greift die Routine nun ein und führt die Formatierung durch. Anschließend wird der formatierte String dann ausgegeben.

Der Aufruf über eine SYS-Funktion hat folgendes Format:

```
SYS (AD) X
```

Dabei ist AD die Startadresse der Routine und X die auszudruckende Variable oder ein positiver numerischer Ausdruck. Die Parameter für den Ausdruck werden mit POKES gesetzt:

```

POKE 51612, X 0 = ganze Zahl, 1 = Zahl mit Dezimalpunkt
POKE 51613, L 0 - 10, Gesamtlänge - 1
POKE 51614, N Anzahl der Stellen nach dem Dezimalpunkt
POKE 51615, ASC(" ") Füllzeichen vor der Zahl
POKE 51549, ASC(" ") Führendes Zeichen vor der Zahl

```

Die vorgewählten Werte sind: Dezimalzahl, Länge 10 (9+1), 2 Dezimalstellen, Leerzeichen als führendes Zeichen und als Füller.

SYS (51456) 100 führt zu folgendem Ausdruck:

```
100.00
```

Mit POKE 51614,3 : POKE 51615,ASC("#") :POKE 51549,ASC("\$") sieht die Ausgabe des obigen Beispiels so aus:

```
##$100.000
```

Wollen Sie die Ausgabe auch auf dem Drucker haben, müssen Sie den CMD-Modus benutzen:

```

OPEN 1,4 : CMD 1
SYS (51456) X
PRINT# 1 : CLOSE 1

```

```

; PRINT USING
C900 20 8A AD      JSR FRMNUM ; numerischen Ausdruck holen
C903 20 DD 0D      JSR ASCII  ; nach ASCII umwandeln
C906 20 0D C9      JSR USING
C909 20 1E AB      JSR OUT   ; String ausgeben
C90C 60            RTS
C90D A9 45        LDA #E"
C90F 20 8E C9      JSR CHECK  ; prüft auf Exponentialdarstellung

```

C912 B0 59	BCS	SETPTR	
C914 AD 9C C9	LDA	DECINT	; Flag für dezimal oder ganze Zahl
C917 F0 59	BEQ	INTEGER	
C919 AD 02 01	LDA	\$102	
C91C D0 0B	BNE	L1	
C91E AC 90 C9	LDY	LENGTH	; Gesamtlänge - 1
C921 A9 30	LDA	0"0"	
C923 99 02 01 L2	STA	\$102,Y	; Puffer mit Nullen füllen
C926 B8	DEY		
C927 D0 FA	BNE	L2	
C929 A9 2E L1	LDA	0". "	
C92B 20 BE C9	JSR	CHECK	
C92E A8	TAY		
C92F 90 02	BCC	+ + 4	
C931 A0 30	LDY	0"0"	
C933 A9 00	LDA	00	
C935 20 BE C9	JSR	CHECK	
C938 98	TYA		
C939 9D 00 01	STA	\$100,X	
C93C A9 2E	LDA	0". "	
C93E 20 BE C9	JSR	CHECK	
C941 AC 9E C9	LDY	DECLEN	; Anzahl der Dezimalstellen
C944 E8 L3	INX		
C945 B8	DEY		
C946 D0 FC	BNE	L3	
C948 EC 9D C9 L8	CPX	LENGTH	
C94B B0 20	BCS	SETPTR	
C94D AC 9D C9	LDY	LENGTH	
C950 A9 00	LDA	00	
C952 99 01 01	STA	\$101,Y	
C955 BD 00 01 L6	LDA	\$100,X	
C958 C9 20	CHP	0" "	; führendes Zeichen
C95A D0 02	BNE	L5	
C95C A9 20	LDA	0" "	
C95E 99 00 01 L5	STA	\$100,Y	
C961 CA	DEX		
C962 10 06	BPL	L4	
C964 AD 9F C9	LDA	FILLER	
C967 B8	DEY		
C96B 10 F4	BPL	L5	
C96A B8 L4	DEY		
C96B 10 E8	BPL	L6	
C96D A9 00 SETPTR	LDA	00	; Zeiger auf Puffer setzen
C96F A0 01	LDY	01	
C971 60	RTS		
C972 A9 00 INTEGER	LDA	00	
C974 20 BE C9	JSR	CHECK	
C977 90 F4	BCC	SETPTR	
C979 BA	TXA		
C97A A8	TAY		
C97B AD 02 01	LDA	\$102	
C97E F0 09	BEQ	L7	
C980 A9 2E	LDA	0". "	
C982 20 BE C9	JSR	CHECK	
C985 90 02	BCC	L7	
C987 BA	TXA		
C98B A8	TAY		
C989 98 L7	TYA		
C98A AA	TAX		
C98B CA	DEX		
C98C 10 BA	BPL	L8	



```

C98E A2 00      CHECK      LDY  #0
C990 DD 00 01 L9          CMP  $100,X
C993 F0 06              BEQ  L10
C995 E8              INCY
C996 E0 0C              CPY  #12
C998 D0 F6              BNE  L9
C99A 18              CLC
C99B 60              RTS
C99C 01      DECINT      .BYT 1      ; dezimal
C99D 09      LENGTH     .BYT 9      ; Länge 9
C99E 02      DECLN      .BYT 2      ; Anzahl der Dezimalstellen
C99F 20      FILLER     .BYT " "    ; Füllzeichen
C9A0              LEADING = L5 - 1 ; führendes Zeichen

```

Hier das entsprechende Ladeprogramm in BASIC.

```

100 FOR I = 51456 TO 51615
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 32,138,173, 32,221,189, 32, 13,201, 32, 30,171
130 DATA 96,169, 69, 32,142,201,176, 89,173,156,201,240
140 DATA 89,173, 2, 1,208, 11,172,157,201,169, 48,153
150 DATA 2, 1,136,208,250,169, 46, 32,142,201,168,144
160 DATA 2,160, 48,169, 0, 32,142,201,152,157, 0, 1
170 DATA 169, 46, 32,142,201,172,158,201,232,136,208,252
180 DATA 236,157,201,176, 32,172,157,201,169, 0,153, 1
190 DATA 1,189, 0, 1,201, 32,208, 2,169, 32,153, 0
200 DATA 1,202, 16, 6,173,159,201,136, 16,244,136, 16
210 DATA 232,169, 0,160, 1, 96,169, 0, 32,142,201,144
220 DATA 244,138,168,173, 2, 1,240, 9,169, 46, 32,142
230 DATA 201,144, 2,138,168,152,170,202, 16,186,162, 0
240 DATA 221, 0, 1,240, 6,232,224, 12,208,246, 24, 96
250 DATA 0, 9, 2, 32
260 IF S <> 18657 THEN PRINT "FEHLER IN DATAS !!" : END
270 PRINT "OK !"

```

### 5.3.3 Mathematische Routinen selbst entwickelt

Wenn wir oft mathematische Routinen benötigen, die der Interpreter nicht bietet, so lohnt es sich, dafür selber ein Unterprogramm in Maschinensprache zu schreiben. Für Funktionen mit einem Argument bietet sich dafür die USR-Funktion an. Wie funktioniert nun die USR-Funktion? Sie kann genauso wie alle anderen Funktionsaufrufe des Interpreters, z.B. wie die SIN-Funktion in Ausdrücken zur Berechnung von Variablen oder auch in einem PRINT-Statement stehen. Damit der Interpreter weiß, wo die eigene Routine steht, wird ihm noch in zwei Speicherzellen die Startadresse der Routine mitgeteilt. Dies kann im BASIC mit POKE-Befehlen geschehen. Beim Aufruf der USR-Funktion wird der Wert des Arguments (das kann ein beliebiger komplizierter Ausdruck sein) im Fließkommakku 1 FAC übergeben. Jetzt kann die Funktion berechnet werden. Wenn das Ergebnis im FAC steht und RTS (RETURN vom Maschinenprogramm) ausgeführt wird, so wird der Wert wieder an BASIC übergeben. Sehen wir uns jetzt an ein paar Beispielen an, wie man eigene Funktionen schreibt.

### 5.3.4 SQR-, SUM- und PROD-Funktion

Zuerst wollen wir uns eine Routine zur Berechnung der Quadratwurzel schreiben. Der BASIC-Interpreter stellt eine solche Funktion zwar bereits zur Verfügung, unsere Routine soll jedoch schneller und genauer werden, da wir nicht wie der Interpreter eine Potenzierung durchführen (die den Aufruf von LOG und EXP erfordert), sondern eine Iteration durchführen. Als Startwert nehmen wir dazu das Argument und halbieren den Exponent, was bereits eine gute Schätzung des Wurzelwertes ist. Die Iterationsvorschrift lautet  $x(n+1) = (x(n) + a/x(n)) / 2$ , wobei  $a$  das Argument und  $x(n)$  und  $x(n+1)$  der alte und der neue Schätzwert sind. Durch Ausprobieren zeigt sich, daß sich das Ergebnis nach 4 Iterationen nicht mehr ändert.

```

C800 20 2B BC      JSR SIGN ; Vorzeichen testen
C803 F0 34        BEQ ENDE ; Wert gleich 0, fertig
C805 10 03        BPL OK   ; positiv, dann in Ordnung
C807 4C 48 B2     JMP ILL   ; negativ, 'illegal quantity'
C80A 20 C7 BB OK   JSR FACA4 ; FAC nach Akku#4 übertragen
C80D A5 61        LDA EXP
C80F 38          SEC
C810 E9 81        SBC #5B1 ; Exponent normalisieren
C812 0B          PHP
C813 4A          LSR A      ; Exponent halbieren
C814 18          CLC
C815 69 01        ADC #1
C817 28          PLP
C818 90 02        BCC S1
C81A 69 7F        ADC #7F ; Exponent wieder herstellen
C81C 85 61 S1     STA EXP
C81E A9 04        LDA #4   ; 4 Iterationen
C820 B5 67        STA COUNT
C822 20 CA BB ITER JSR FACA3 ; FAC nach Akku#3
C825 A9 5C        LDA #5C
C827 A0 00        LDY #00 ; Zeiger auf Akku#4
C829 20 0F BB     JSR DIV   ; durch FAC dividieren
C82C A9 57        LDA #57
C82E A0 00        LDY #00 ; Zeiger auf Akku#3
C830 20 67 BB     JSR PLUS ; zu FAC addieren
C833 C6 61        DEC EXP ; FAC / 2 (Exponent minus 1)
C835 C6 67        DEC COUNT ; Zähler erniedrigen
C837 D0 E9        BNE ITER ; noch eine Iteration
C839 60          ENDE RTS ; fertig

```

Bevor wir unsere neue USR-Funktion aufrufen, müssen wir dem Interpreter erst mitteilen, wo unsere USR-Funktion beginnt. Dazu wird das low-Byte der Adresse nach \$311 (dezimal 785) und das high Byte nach \$312 (786) gepoket. Für unsere Funktion sähe das so aus:

```
POKE 785,0 : POKE 786, 12*16+8
```

Das folgende Ladeprogramm in BASIC enthält diese Pokes bereits.

```

100 FOR I = 51200 TO 51257
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 32, 43,188,240, 32, 16, 3, 76, 72,178, 32,199

```

```

130 DATA 187,165, 97, 56,233,129, 8, 74, 24,105, 1, 40
140 DATA 144, 2,105,127,133, 97,169, 4,133,103, 32,202
150 DATA 187,169, 92,160, 0, 32, 15,187,169, 87,160, 0
160 DATA 32,103,184,198, 97,198,103,208,233, 96
170 IF S <> 6211 THEN PRINT "FEHLER IN DATAS !!" : END
180 POKE 785,0 : POKE 786,200 : PRINT "OK !"

```

Jetzt läßt sich mit ? USR(A) unsere Routine aufrufen. Vergleicht man die Ausführungszeit unserer Routine mit der SQR-Routine des Interpreters, so ist unsere mit ca. 12 Millisekunden gegenüber ca. 52 Millisekunden etwa 4 mal so schnell wie die des Interpreters.

Jetzt wollen wir uns noch ein etwas komplizierteres Beispiel ansehen.

Oft steht man vor der Aufgabe, eine Zahlenreihe zu addieren. Dies ist z.B. bei der Ermittlung des Durchschnitts oder anderen statistischen Berechnungen der Fall. Wir nehmen an, daß die Daten in einem Array (dimensionierte Variable) zur Verfügung stehen.

```

10 DIM A(1000)
  . . . Berechnung oder Einlesen der Daten
100 S = 0
110 FOR I = 0 TO 1000 : S = S + A(I) : NEXT
120 PRINT S

```

Unsere USR-Funktion soll nun die BASIC-Zeilen 100 und 110 ersetzen, wir wollen dafür

```
100 S = USR(A)
```

schreiben. Der Parameter A steht dabei für den Arraynamen. Wie wir später sehen werden, läßt sich durch Ändern zweier Maschinenbefehle auch das Produkt der Arrayelemente berechnen.

033C 20 BD AD	JSR	NUMTEST ; Variable numerisch ?
033F A6 2F	LDX	ARRTAB
0341 A5 30	LDA	ARRTAB+1 ; Zeiger auf Beginn Arraytabelle
0343 B6 5F	STX	TEMP
0345 B5 60	STA	TEMP+1 ; laufender Zeiger
0347 C5 32	CMP	ARREND+1
0349 D0 04	BNE	S1
034B E4 31	CPX	ARREND ; Ende der Arraytabelle ?
034D F0 1D	BEQ	NOTFOUND
034F A0 00	LDY	#0
0351 B1 5F	LDA	(TEMP),Y ; erster Buchstabe des Namens
0353 C8	INY	
0354 C5 45	CMP	VARNAM ; mit gesuchtem Namen vergleichen
0356 D0 06	BNE	S2 ; nein, dann nächstes Array testen
0358 A5 46	LDA	VARNAM+1 ; zweiter Buchstabe
035A D1 5F	CMP	(TEMP),Y ; vergleichen
035C F0 17	BEQ	FOUND ; gefunden
035E C8	INY	
035F B1 5F	LDA	(TEMP),Y
0361 18	CLC	
0362 65 5F	ADC	TEMP ; Offset für nächstes Array addieren
0364 AA	TAX	
0365 C8	INY	

0366	B1	5F		LDA	(TEMP),Y	
0368	65	60		ADC	TEMP+1	
036A	90	D7		BCC	S3	
036C	A2	E2	NOTFOUND	LDX	#< TAB	
036E	B6	22		STX	\$22	; Zeiger auf Fehlermeldung
0370	A7	03		LDA	#> TAB	
0372	4C	45	##	JMP	ERROUT	; Fehlermeldung ausgeben
0375	CB		FOUND	INX		
0376	B1	5F		LDA	(TEMP),Y	
0378	18			CLC		
0379	65	5F		ADC	TEMP	
037B	85	24		STA	STORE	
037D	CB			INX		
037E	B1	5F		LDA	(TEMP),Y	
0380	65	60		ADC	TEMP+1	
0382	85	25		STA	STORE+1	
0384	CB			INX		
0385	B1	5F		LDA	(TEMP),Y	; Anzahl der Indizes
0387	20	96	B1	JSR	SETARR	; Zeiger auf erstes Arrayelement
038A	85	5F		STA	TEMP	
038C	B4	60		STY	TEMP+1	; Zeiger nach Temp
038E	24	0E		BIT	INTFLAG	; Integerflag testen
0390	30	1F		BMI	INTEGER	
0392	20	A2	BB	JSR	MEMAC1	; Element in FAC
0395	18			CLC		
0396	90	04		BCC	LOOP	; Sprung in Schleife
0398	20	67	BB S5	JSR	MEMPLUS	; Variable plus FAC
039B	18			CLC		
039C	A5	5F	LOOP	LDA	TEMP	
039E	67	05		ADC	#5	; Zeiger auf nachstes Element
03A0	85	5F		STA	TEMP	
03A2	90	02		BCC	S4	
03A4	E6	60		INC	TEMP+1	
03A6	A4	60	S4	LDY	TEMP+1	
03A8	C5	24		CMP	STORE	; Ende des Arrays ?
03AA	90	EC		BCC	S5	
03AC	C4	25		CPY	STORE+1	
03AE	90	EB		BCC	S5	
03B0	60		READY	RTS		; ja, fertig
03B1	20	D5	03 INTEGER	JSR	INTAKK	; Integervariable nach FAC
03B4	20	0C	BC S6	JSR	A1DA2	; FAC nach ARG
03B7	18			CLC		
03B8	A5	5F		LDA	TEMP	
03BA	69	02		ADC	#2	; Zeiger auf nächstes Arrayelement
03BC	85	5F		STA	TEMP	
03BE	90	02		BCC	S7	
03C0	E6	60		INC	TEMP+1	
03C2	C5	24	S7	CMP	STORE	; Ende des Arraybereichs ?
03C4	90	06		BCC	S8	
03C6	A5	60		LDA	TEMP+1	
03C8	C5	25		CMP	STORE+1	
03CA	B0	E4		BCC	READY	
03CC	20	D3	03 S8	JSR	INTAKK	; Integervariable nach FAC holen
03CF	20	6F	BB	JSR	AKPLUS	; FAC + ARG
03D2	4C	B4	03	JMP	S6	
03D5	A0	00	INTAKK	LDY	#0	
03D7	B1	5F		LDA	(TEMP),Y	
03D9	AA			TAX		
03DA	CB			INX		
03DB	B1	5F		LDA	(TEMP),Y	
03DD	AB			TAY		

```

03DE BA          TIA
03DF 4C 91 B5     JMP INTFLOAT ; nach Fließkomma
03E2 41 52 52 TAB .ASC "ARRAY NOT FOUND"
03E5 41 59 20 4E 4F 54 20 46 50 55 4E
03F0 C4          .BYTE "D" + $80

```

```

100 FOR I = 828 TO 1008
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 32,141,173,166, 47,165, 48,134, 95,133, 96,197
130 DATA 50,208, 4,228, 49,240, 29,160, 0,177, 95,200
140 DATA 197, 69,208, 6,165, 70,209, 95,240, 23,200,177
150 DATA 95, 24,101, 95,170,200,177, 95,101, 96,144,215
160 DATA 162,226,134, 34,169, 3, 76, 69,164,200,177, 95
170 DATA 24,101, 95,133, 36,200,177, 95,101, 96,133, 37
180 DATA 200,177, 95, 32,150,177,133, 95,132, 96, 36, 14
190 DATA 48, 31, 32,162,187, 24,144, 4, 32,103,184, 24
200 DATA 165, 95,105, 5,133, 95,144, 2,230, 96,164, 96
210 DATA 197, 36,144,236,196, 37,144,232, 96, 32,213, 3
220 DATA 32, 12,188, 24,165, 95,105, 2,133, 95,144, 2
230 DATA 230, 96,197, 36,144, 6,165, 96,197, 37,176,228
240 DATA 32,213, 3, 32,111,184, 76,180, 3,160, 0,177
250 DATA 95,170,200,177, 95,168,138, 76,145,179, 65, 82
260 DATA 82, 65, 89, 32, 78, 79, 84, 32, 70, 79, 85, 78
270 DATA 196
280 IF S <> 20399 THEN PRINT "FEHLER IN DATAS !!" : END
290 POKE 785, 3*16+12 : POKE 786, 3 : PRINT "OK !"

```

Das Programm kann sowohl Arrays mit reellen Zahlen als auch Integer-Arrays verarbeiten. Wird ein Array nicht gefunden, so wird die Fehlermeldung 'array not found error' ausgegeben. Da die Logik zum Errechnen des Produkts der Arrayelemente gleich ist, kann man eine Produktfunktion erhalten, indem wir die Aufrufe zur Addition durch die Multiplikationsroutine ersetzen.

Dazu muß ab Adresse \$0398 20 28 BA stehen und ab Adresse \$03CF steht 20 28 BA. Von BASIC aus kann dies mit POKE 921,40 : POKE 922, 186 : POKE 976, 43 : POKE 977, 186 geschehen.

Um unsere Routine, die diesmal im Bandpuffer liegt, benutzen zu können, müssen wir wieder die Startadresse poken (im BASIC-Lader schon geschehen):

```
POKE 785, 3*16+12 : POKE 786, 3
```

Berechnen Sie zum Vergleich einmal die Summe mit einer BASIC-Schleife und dann mit unserer Routine - der Zeitunterschied ist gewaltig.

### 5.3.5 Umwandlung der verschiedenen Datenformate

Sollen mehr als ein Parameter übergeben werden, so ist die USR-Funktion nicht mehr geeignet. Hier bietet sich eine erweiterte Variante des SYS-Befehls an. Normalerweise führt der SYS-Befehl nur das Maschinenprogramm ab dieser Adresse durch und übergibt keine weiteren Parameter. Die oben angesprochene Routine FRMEVL liest jedoch einen folgenden Parameter und übergibt ihn im Fließkommakku. Durch Komma oder Klammern getrennt können so beliebig viele Parameter

übergeben werden. Zu der Routine zur Formelauswertung stehen noch eine Reihe weiterer Einsprungpunkte und Unterroutinen zur Verfügung, die z.B. Parameter in Klammer auswerten, auf ein nachfolgendes Komma prüfen. Auch läßt sich der Variablentyp - String oder numerisch - testen. Bei numerischen Variablen ist zusätzlich noch eine Bereichsüberprüfung möglich. Die wichtigsten Routinen sind unten zusammengestellt. Weitere Einzelheiten entnehmen Sie bitte wieder unserem ROM-Listing oder Kapitel 6.4.

#### Adresse Beschreibung

ADBA	Argument auswerten und auf numerisch prüfen
ADBD	auf numerisch prüfen
ADBF	auf String prüfen
AD9E	Argumentauswertung, beliebiger Ausdruck
AEF1	Argument in Klammern auswerten
AEF7	prüft auf Klammer zu
AEFA	prüft auf Klammer auf
AEFD	prüft auf Komma
B79E	holt Byte, (0 bis 255) in X-Register
0073	holt nächstes Zeichen aus BASIC-Text

Bei Bereichsüberschreitung wird 'illegal quantity' ausgegeben, falscher Typ ergibt 'type mismatch'.

Die Umwandlung der verschiedenen Formate ineinander ist mit folgenden Routinen möglich:

#### Adresse Beschreibung

B1BF	wandelt FAC nach Integer
B395	wandelt 16-Bit Integerzahl in A/X nach Fließkomma
B3A2	wandelt Byte in Y nach Fließkomma
BC9B	wandelt FAC nach 16-Bit Zahl
BCF3	wandelt Ziffernstring nach Fließkomma
BDDD	wandelt FAC in Ziffernstring

Jetzt wollen wir uns noch ein Beispiel für einen SYS-Aufruf mit Parameterübergabe ansehen.

Will man von BASIC aus eine Bildschirmausgabe an eine bestimmte Position machen, so muß man mit der Cursorsteuerung nach HOME die entsprechende Anzahl an Cursor right und Cursor down Zeichen drucken. Dies ist umständlich und speicherplatzaufwendig. Einfacher geht es mit einer selbstgeschriebenen Maschinenroutine.

Der Aufruf soll folgende Syntax haben:

SYS PR, Spalte, Zeile, Druckliste

Dabei ist PR die Startadresse der Routine, Zeile und Spalte sind die Cursorposition, an die die Variablen oder Ausdrücke der Druckliste wie beim normalen PRINT-Befehl ausgegeben werden.

C000	20	FD	AE	JSR	CKCOM	; prüft auf Komma
C003	20	9E	B7	JSR	GETBYT	; holt Spaltenwert nach X
C006	8A			TXA		
C007	48			PHA		; Spaltennummer merken

```

C00B 20 FD AE      JSR  CKCOM  ; prüft auf Komma
C00B 20 9E B7      JSR  GETBYT ; holt Zeilenwert
C00E 6B           PLA           ;
C00F AB           TAY           ; Spaltenwert nach Y
C010 1B           CLC
C011 20 F0 FF      JSR  CURSOR ; setzt Cursor
C014 20 FD AE      JSR  CKCOM  ; prüft auf Komma
C017 4C A4 AA      JMP  PRINT  ; weiter mit PRINT-Befehl

```

Hier wieder ein kurzes Ladeprogramm:

```

100 FOR I = 49152 TO 49177
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 32,253,174, 32,158,183,138, 72, 32,253,174, 32
130 DATA 158,183,104,168, 24, 32,240,255, 32,253,174, 76
140 DATA 164,170
150 IF S <> 3566 THEN PRINT "FEHLER IN DATAS !!" : END
160 PRINT "OK !"

```

Wenn man zu Anfang des Programms der Variablen PR die Startadresse \$C000 der Routine zuweist, läßt sich mit dem folgenden Befehl der Text "Beispiel" ab der 24. Spalte der 20. Zeile ausgeben.

```

10 PR = 12*4096
100 SYS PR, 24, 20, "Beispiel"

```

## 6.1 Der Monitor - Und was steckt dahinter ?

Zur Programmierung in Maschinsprache benötigen Sie sinnvollerweise einen sogenannten MONITOR (nicht zu verwechseln mit einem Datensichtgerät oder Fernseher). Der Monitor ist ein Hilfsprogramm. Er ermöglicht das Verändern von Speicherplätzen und Registern, die Ausführung von Maschinenprogrammen und das Laden, Abspeichern und Disassemblieren von Maschinenprogrammen. Außerdem kann man mit dem Monitor schon erstellte Maschinenprogramme leicht ändern und korrigieren.

Wir wollen in diesem Kapitel nun zeigen, wie man mit einem normalen Monitor arbeitet.

Nachdem der Monitor von Kassette oder Diskette gelesen wurde, startet man ihn mit dem üblichen Befehl zum Aufruf von Maschinenprogrammen. Dieser Befehl lautet:

SYS xxxxx

Bei unserem Monitor würde man sagen: SYS 12\*4096. Diese Adresse ist die Startadresse des Monitors in dezimal. Hexadezimal ist der Wert C000. Nach diesem Aufruf meldet sich der Monitor mit seinem PROMPT (Meldung, daß er nun Monitor-spezifische Befehle entgegennehmen kann). In unserem Fall sieht die Meldung so aus:

```
C>      PC  IRQ  SR AC XR YR SP
>; E145 EA13 31 32 AC 34 F8
>
```

Diese Angaben müssen nun erst einmal genauer erklärt werden. Dazu müssen wir uns kurz mit den REGISTERN des 6502 (6510) beschäftigen. Denn diese Register werden hier angezeigt. Im folgenden bedeutet daher:

Der Programnzähler: PC

In diesem Register steht immer die nächste zu Verarbeitende Speicherstelle. In unserem Beispiel ist das die Adresse E145.

Der Interruptvektor: IRQ

Dieses Register steuert eine mögliche Programmunterbrechung und Verzweigung.

Das Statusregister: SR

In diesem Register befinden sich die Zustände der FLAGGEN (Flags). Wir kennen hier folgende Flaggen:

- b7 = Negativ Flagge (Gesetzt bei negativem Ergebnis)
- b6 = Interner Überlauf (Gesetzt bei arithm. Überlauf)
- b5 = IMMER GESETZT
- b4 = Abbruch Flagge (Gesetzt bei Programmabbruch)
- b3 = Dezimal Flagge (Wechselt von Hex. nach BCD Arithm.)
- b2 = Interrupt Schalter (Steuert den Interrupt)
- b1 = Null-Flagge (Gesetzt wenn Ergebnis = 0)
- b0 = Übertrag Flagge (Gesetzt bei arithm. Übertrag)

In den meisten Handbüchern finden Sie folgende Darstellung:



b7 b6 b5 b4 b3 b2 b1 b0  
N V 1 B D I Z C

Der Akkumulator: AC

In diesem Register werden alle Vergleichs- und Rechenoperationen ausgeführt.

Das X-Register: XR

Dieses Indexregister dient zur Aufnahme von Daten während des Programmablaufs.

Das Y-Register: YR

Dieses Register hat den selben Zweck wie das oben beschriebene X-Register.

Der Stapelzeiger: SP

Dieses Register zeigt auf den Stapel, in dem die Rücksprungadressen der Unterprogramme abgelegt sind.

Die Inhalte dieser Register können laufend überprüft und verändert werden. Zu beachten ist, daß alle Eingaben von Zahlen in hexadezimaler Form zu erfolgen haben. Also zum Beispiel FF für die Zahl 255. Zum Ändern der Registerinhalte wird einfach der CURSOR (das blinkende Kästchen auf dem Bildschirm) über den entsprechenden Wert gebracht, dies geschieht durch Betätigung der Pfeil-Tasten auf der Commodore 64 Tastatur, und dann der neue Wert eingegeben. Durch Drücken der RETURN-Taste wird dann der neue Registerinhalt übernommen.

Genau so, wie Sie die Registerinhalte anzeigen und ändern, können Sie auch die einzelnen Speicherinhalte behandeln. Dies bedeutet dann die eigentliche Programmierung in Maschinensprache. Um einen bestimmten Speicherbereich anzuzeigen, gibt es folgenden Befehl:

M xxxx yyyy

Das M bedeutet MEMORY, also Speicher, xxxx ist die Anfangs- und yyyy die Endadresse des auszugebenden Speicherbereichs. Alle Adressen müssen als 4-stellige Hexadezimalzahlen angegeben werden (z.B. 02A1 für 673)!

Nach dieser Eingabe werden dann die entsprechenden Speicherinhalte angezeigt. Wenn der angegebene Speicherbereich nicht vollständig auf den Bildschirm geht, verschiebt sich der Bildschirm Zeile für Zeile nach oben, bis die Endadresse ausgegeben wurde. Diesen Vorgang kann man durch Drücken der STOP-Taste abbrechen. Man befindet sich dann wieder in der Kommandozeile.

So wie wir eben die Registerinhalte verändert haben, so lassen sich auch jetzt die Speicherinhalte verändern. Man fährt also mit dem Cursor wieder an die entsprechende Stelle, gibt dann den neuen Wert, oder die neuen Werte ein, und drückt dann die RETURN-Taste.

So könnte zum Beispiel eine Ausgabe von einem bestimmten

Speicherbereich aussehen:

```
>M C000 C010
>: C000 A9 10 BD 16 03 A9 C0 BD
>: C008 17 03 A9 43 85 97 D0 16
>: C010 A9 42 85 97 D8 4A 68 BD usw.
```

Der Speicher kann aber auch auf eine andere Art und Weise dargestellt werden: Durch das DISASSEMBLIEREN von Maschinenprogrammen.

Das Disassemblieren hat den Vorteil, daß aus den unübersichtlichen Hexzahlen, leicht zu verstehende Befehlsfolgen werden. Diese Befehlsfolgen bezeichnet man allgemein auch als MNEMONICS. Die recht aufwendige Übersetzung der Hexzahlen in "Klartext", die man sonst von Hand aus erledigen muß, entfällt beim Disassemblieren. Allerdings hat das Disassemblieren auch einen Nachteil:

Wenn irgendwo in dem Maschinenprogramm zum Beispiel ein Text, der also überhaupt nichts mit Maschinenbefehlen zu tun hat, erscheint, versucht der Disassembler diesen Befehl als OPERATIONSCODE zu interpretieren, um aus diesem dann die Mnemonics zu machen. Bei nicht existierenden Opcodes, die aus einer solch fehlerhaften Interpretation entstehen können, werden an Stelle der Mnemonics nur Fragezeichen ausgegeben. Es kann aber auch passieren, daß eine anscheinend normale Mnemonics entsteht. Jeder erfahrene Maschinenprogrammierer wird aber rasch feststellen, was Text oder Variablen sind, und was Maschinenbefehle sind.

Der nächste Schritt nach Eingabe oder Änderung eines Maschinenprogramms ist die Programmausführung. Dies geschieht mit diesem Befehl:

G xxxx

Das G steht hier für das englische Wort GO TO und bedeutet einen Sprung an die angegebene Adresse xxxx. Diese Adresse ist entweder die Startadresse des Maschinenprogramms oder eine Einsprungadresse aus irgendeiner Maschinenroutine.

Trifft das Maschinenprogramm während seiner Ausführung auf einen BRK-(BREAK)-Befehl, der im übrigen dem STOP-Befehl in BASIC-Programmen entspricht, so wird die Ausführung abgebrochen, und der Monitor meldet sich wieder mit B\*. Durch Einfügen dieses Befehls in Maschinenprogrammen, ist es möglich, dieses Programm schnell und einfach auszutesten. Nach dem Austesten des Programmes kann der BRK-Befehl gegen den eigentlichen Befehl, der vor dem Einsetzen von BRK an dieser Speicherstelle stand, wieder ausgetauscht werden.

Maschinenprogramme sollen natürlich nicht nur getestet und verändert werden, wir wollen Sie auch auf Diskette oder Kassette abspeichern. Der Monitor hat für diesen Vorgang den entsprechenden Befehl. Er lautet:

S "NAME",xx,yyyy,zzzz

Das S bedeutet SAVE (speichern), an Stelle von NAME können Sie einen beliebigen Namen für das Programm angeben (die Anführungszeichen "" müssen mit angegeben werden), xx bedeutet

die Geräteadresse (01 = Kassette, 08 = Diskette), yyyy ist die Anfangs- und zzzz die Endadresse des abzuspeichernden Maschinenprogramms.

Nach Drucken der RETURN-Taste wird dann das Programm auf dem angegebenen Gerät abgespeichert.

Das Laden von Maschinenprogrammen erfolgt nach einem ähnlichen Prinzip. Hier lautet der Befehl:

L "NAME",xx

Er unterscheidet sich in diesem ersten Teil nur durch das L für LOAD (oder Laden). Einige Monitore bieten zusätzlich die Möglichkeit Maschinenprogramme an verschiedene Speicherstellen zu laden. So können Sie ein Programm, das bei der Adresse 0800 beginnt an die Adresse 2800 legen. Aber Vorsicht ! Es kann passieren, daß das verschobene Programm nicht lauffähig ist. Es müssen dann noch alle absoluten Adressen geändert werden.

Als Abschluß der Programmierung im Monitor gibt es den X-Befehl. X bedeutet hierbei EXIT (Ausgang). Durch Verwendung dieses Befehls kommt man wieder ins BASIC zurück. Es ist hierbei von Vorteil, daß weder das Maschinenprogramm, noch ein mögliches BASIC-Programm zerstört wurde. Man muß jedoch darauf achten, daß das Maschinenprogramm nicht an die selbe Adresse des BASIC-Programms gelegt wurde. Dann würde nämlich das BASIC-Programm überschrieben werden.

Ein Tip am Rande: Wenn Sie ein Maschinenprogramm geladen haben, geben Sie als ersten Befehl, noch vor dem Starten des Maschinenprogrammes, NEW ein. Dieser Befehl setzt wieder alle Zeiger auf ihren normalen Zustand zurück. Es könnte sonst passieren, daß Sie bei Eingabe eines BASIC-Programms, schon nach der ersten Zeile, eine seltsame Fehlermeldung bekommen.

Natürlich kann man auch ohne Monitor in Maschinensprache programmieren. Zumindest kann man die Maschinenprogramme eingeben und starten. Doch muß man dabei beachten, daß jede einzelne Speicherstelle von BASIC aus geändert werden muß.

Das bedeutet, daß Sie zunächst ein BASIC-Programm schreiben müßten, mit dem Sie die Operationscodes eingeben können. Das ist aber nicht sehr einfach, da Sie alle Operationscodes bei der Eingabe in dezimaler Form angeben müssen, weil der dazu notwendige POKE-Befehl nur dezimale Zahlen erlaubt, oder aber eine zusätzliche dezimal-hex Umwandlungsroutine schreiben müssen. Bei der Ausgabe der Bytes tritt dann das genau entgegengesetzte Problem auf, da dann die mit PEEK gelesenen Werte wiederum in dezimaler Form sind.

Wie Sie sehen, ist die Anschaffung eines in der Regel recht preiswerten Monitors jedem angehenden Maschinenprogrammierer, oder dem der es werden will, aber auch dem 'Nur-Hobbyisten', unbedingt zu empfehlen.

## 6.2 Nützliche Adressen des Commodore 64 Betriebssystems

Wenn man eigene Programme in Maschinensprache schreibt, kann man sich durch geschickte Ausnutzung der ROM-Routinen viel Arbeit sparen. Besonders die Routinen zur Bedienung der Peripheriegeräte bieten sich dazu an.

Die wichtigsten Routinen des Commodore 64 sind am Ende des ROMs als Sprungtabelle auf die eigentlichen Routinen zusammengefaßt. Diese Adressen wurden beim Erscheinen vom neuen Commodore Rechnern nicht geändert, sondern nur erweitert. Deshalb es ist zum Beispiel möglich Routinen, die für einen großen CBM-Rechner geschrieben wurden, ohne Schwierigkeiten auf dem Commodore 64 zu übernehmen, sofern nur diese sogenannten 'Kernal'-Routinen benutzt wurden. Die Sprungtabelle des Commodore 64 ist bis auf drei zusätzliche Adressen mit der des VC 20 identisch, sodaß es mit Hilfe dieser Routinen ein leichtes ist, Programme des VC 20 zu konvertieren. Wir wollen uns jetzt einige dieser Routinen etwas näher ansehen.

Adresse	Funktion
\$FF90	setzt Flag für Ausgabe von Systemmeldungen
\$FF93	schickt Sekundaradresse nach einem LISTEN-Befehl auf den IEC-Bus
\$FF96	schickt Sekundaradresse nach einem TALK-Befehl auf den IEC-Bus
\$FF99	holt bei gesetztem Carry Flag die höchste RAM-Adresse nach X und Y, bei geloschtem Carry-Flag wird die Adresse von X und Y gesetzt.
\$FF9C	die selbe Funktion wie oben, jedoch für den RAM-Anfang
\$FF9F	fragt die Tastatur ab
\$FFA2	setzt das Time-out Flag für den IEC-Bus
\$FFA5	holt ein Byte vom IEC-Bus in den Akku
\$FFA8	gibt ein Byte aus dem Akku an den IEC-Bus aus
\$FFAB	sendet UNTALK-Befehl auf den IEC-Bus
\$FFAE	sendet UNLISTEN-Befehl auf den IEC-Bus
\$FFB1	sendet LISTEN-Befehl auf den IEC-Bus
\$FFB4	sendet TALK-Befehl zum IEC-Bus
\$FFB7	holt das Statuswort in den Akku
\$FFBA	setzt die Fileparameter, Akku muß logischen Filenummer enthalten, X = Gerätenummer und Y = Sekundäradresse

Adresse	Funktion
\$FFBD	setzt Parameter des Filenamens, Akku muß Länge des Namens enthalten, X und Y enthalten die Adresse des Filenamens
\$FFC0	OPEN-Befehl, öffnet logische Datei
\$FFC3	CLOSE-Befehl, schließt logische Datei, Akku muß logische Filenummer enthalten
\$FFC6	CHKIN setzt folgende Eingabe auf logische Datei, die in X übergeben wird Die logische Datei muß vorher mit der OPEN-Routine geöffnet werden
\$FFC9	CKOUT setzt folgende Ausgabe auf logische Datei, die in X übergeben wird Die logische Datei muß vorher mit der OPEN-Routine geöffnet werden
\$FFCC	CLRCH setzt die Ein- und Ausgabe wieder auf Standard (Tastatur/Bildschirm)
\$FFCF	BASIN Eingabe, holt ein Zeichen in den Akku
\$FFD2	BSOUT Ausgabe, gibt Zeichen im Akku aus
\$FFD5	LOAD, lädt Programm in den Speicher
\$FFD8	SAVE, speichert Programm ab
\$FFDB	setzt die laufende Zeit neu
\$FFDE	holt die laufende Zeit
\$FFE1	frägt die STOP-Taste ab
\$FFE4	GET, holt ein Zeichen in den Akku
\$FFE7	CLALL, setzt alle Ein-/Ausgabekanäle zurück, die Dateien werden jedoch nicht geschlossen
\$FFEA	erhöht die laufende Zeit um eine sechzigstel Sekunde
\$FFED	SCREEN holt die Anzahl der Zeilen und und Spalten des Bildschirms
\$FFF0	bei gelöschtem Carry-Flag wird der Cursor auf die Position X/Y gesetzt, bei gesetztem Carry Flag wird die Cursorposition geholt
\$FFF3	holt die Startadresse des I/O-Bausteins

Zur Bedienung des Bildschirm stehen auch einige Routinen zur Verfügung, die Sie als Anwender benutzen können. Dazu gehört auch die schon oben erwähnte Routine zur Positionierung des Cursors. Im folgenden sind die wichtigsten Routinen aufgeführt.

Adresse	Funktion
\$E318	kompletter Reset des Bildschirms und der Tastaturabfrage
\$E544	CLR, löscht den Bildschirm
\$E566	HOME, bringt den Cursor in die linke obere Ecke des Bildschirms
\$E56C	berechnet die Cursorposition
\$E5A0	lädt den Videocontroller mit den Standardwerten
\$E5B4	holt ein Zeichen aus dem Tastaturpuffer
\$E5CA	wartet auf Tastatureingabe
\$E8EA	Bildschirm scrollen, schiebt Bildschirm um eine Zeile nach oben
\$E9FF	löscht eine Bildschirmzeile
\$EA1C	setzt ein Zeichen mit Farbe auf dem Bildschirm (Bildschirmcode im Akku, Farbe in X)

### 6.3. Datenein- und Ausgabe von Maschinenprogrammen aus

Will man eigene Maschinenprogramme schreiben, so kann man besonders für die Datenein- und Ausgabe auf die Routinen des Betriebssystems zurückgreifen. Diese Routinen stehen in einer Sprungtabelle am Ende des ROMS, siehe dazu die letzte Seite des ROM-Listings.

#### 6.3.1. Ein- und Ausgabe von einzelnen Bytes

Die grundlegenden Routinen sind

```
BSOUT  $FFD2  Ausgabe eines Bytes
und    BASIN  $FFCF  Eingabe eines Bytes
```

Das auszugebende bzw. einzulesende Byte wird im Akku übergeben. Der Akku ist das Universalregister des Prozessors, in dem alle Operationen ablaufen.

Beispiel: Ausgabe eines Textes auf den Bildschirm.

```
AUSGABE  LDX  #0
L1       LDA  TEXT,X      ; Text holen
        JSR  BSOUT       ; und ausgeben
        INX
        CPX  #12         ; schon alle Zeichen ?
        BNE  L1
        RTS
TEXT     .ASC  "Beispieltext"
```

Die Eingabe geschieht analog. Soll z.B. ein Text über die Tastatur eingegeben und gespeichert werden, so erscheint der Cursor, und die Zeichen bis zum Drücken der RETURN-Taste werden übernommen.

```
EINGABE  LDX  #0
L1       JSR  BASIN       ; ein Zeichen holen
        STA  TEXT,X      ; und speichern
        INX
        CMP  #13         ; ist es RETURN ?
        BNE  L1         ; nein, weitere Zeichen holen
        RTS
TEXT     =      ...      ; Platz zum Speichern des Textes
```

Diese Routinen geben ein Zeichen auf den Bildschirm aus bzw. holen ein Zeichen von der Tastatur.

Bei der Ausgabe auf den Bildschirm kann natürlich von der Bildschirmsteuerung voll Gebrauch gemacht werden. Dazu gehören zum Beispiel die Codes zur Cursorsteuerung oder zum Bildschirm löschen. Der entsprechende Code wird dazu in den Akku geladen und mit der Ausgaberroutine übergeben.

Beispiel: Bildschirm löschen

```
LDA  #147      ; Kode zum Bildschirm löschen
JSR  BSOUT     ; ausgeben
```

Speziell zur Bildschirmausgabe gibt es noch einige nützliche Routinen, die die Programmierung vereinfachen können.

Die Routine zum Bildschirm löschen kann direkt aufgerufen werden

JSR CLRSCR

Auf für Cursor home existiert eine Routine.

JSR HOME

Besonders interessant ist die Möglichkeit, den Cursor direkt auf eine bestimmte Bildschirmposition zu setzen.

LDX	ZEILE	; Cursorzeile, 0 bis 24
LDY	SPALTE	; Cursorspalte, 0 bis 39
CLC		; Carry clear gleich Cursor setzen
JSR	CURSOR	; Cursor setzen
LDA	"A"	; auszugebendes Zeichen
JSR	BSOUT	; auf Bildschirm

Das Unterprogramm CURSOR hat zwei Funktionen. Bei Aufruf mit geloschtem Carryflag setzt es den Cursor auf die Zeile und Spalte, die im X- und Y-Register stehen. Wird CURSOR dagegen mit gesetztem Carryflag aufgerufen, wird die momentane Cursorposition geholt und im X- und Y-Register übergeben.

Hier die Adressen der obigen Routinen:

CLRSCR	\$E544
HOME	\$E566
CURSOR	\$FFFF

### 6.3.2. Ein- und Ausgabe über Peripheriegeräte

Auf für die Ein- und Ausgabe auf Peripheriegeräte hält das Betriebssystem die benötigten Routinen bereit. Dazu soll kurz auf das Konzept der Ein/Ausgabe eingegangen werden.

Den Peripheriegeräten wird eine Nummer von 0 bis 15 zugewiesen, über die sie vom Betriebssystem angesprochen werden.

Nummer	Gerät
0	Tastatur
1	Datasette
2	RS 232 Schnittstelle
3	Bildschirm
4 - 15	Geräte am seriellen IEC-Bus (Drucker, Floppy)

Zu dieser Geräteadresse oder Primäradresse kommt noch optional eine Sekundäradresse, die die Arbeitsweise des Peripheriegeräts bestimmt und ein Datei- oder 'File'-name. Um nun nicht jedesmal alle Parameter angeben zu müssen, wenn man ein Peripheriegerät anspricht, wird noch die logische Filenummer eingeführt. Zu jeder Filenummer werden einmal mit OPEN die Primär- und Sekundäradresse sowie ein Filename zugeordnet. Jeder weitere Bezug geschieht dann über die logische Filenummer.

Vor der ersten Ein- oder Ausgabe ist die Datei zu eröffnen. Die kann von BASIC aus mit OPEN geschehen oder auch von



Maschinensprache aus. Dazu müssen vorher die Fileparameter gesetzt werden. Die logische Filenummer muß in \$B8 (184) stehen, Gerätenummer in \$BA (186), Sekundärdresse in \$B9 (185), die Länge des Filenamens in \$B7 (183) (Null wenn kein Filename gegeben ist) und die Adresse des Filenamens in \$B8/\$BC (187/188). Dann wird das Unterprogramm OPEN \$FFC0 aufgerufen.

Soll jetzt die Ausgabe auf die geöffnete Datei gehen, so ist folgende Routine aufzurufen:

```
LDX  LF          ; logischen Filenummer
JSR  CKOUT       ; $FFC9, Ausgabe auf Datei legen
```

Wird jetzt die Routine BSOUT (s.o.) aufgerufen, so geht die Ausgabe anstatt auf den Bildschirm auf das Gerät, dem die obige logische Filenummer zugeordnet ist.

Ist LF die logische Filenummer des Druckers, so würde jetzt durch Aufruf des obigen Beispielprogramms AUSGABE der Text auf den Drucker geschrieben.

Die Ausgabe geht solange auf dieses Gerät, bis die Routine CLRCH aufgerufen wird.

```
JSR  CLRCH       ; $FFCC , Ausgabe auf Bildschirm
```

Soll die Dateneingabe aus einer Datei geschehen, z.B. vom Band oder von der Floppy, kann man das folgendermaßen erreichen:

```
LDX  LF          ; Filenummer des Eingabegeräts
JSR  CHKIN       ; $FFC6
```

Jetzt werden durch Aufrufen von BASIN (s.o.) Daten aus der geöffneten Datei geholt. Der Aufruf des Programms EINGABE würde sich jetzt z.B. Daten von der Floppy holen. Dies geschieht solange, bis mit CLRCH wieder auf Standardeingabe (Tastatur) umgeschaltet wird. Die Datei wird dadurch nicht geschlossen. Dies geschieht erst durch Aufruf der Routine CLOSE.

```
LDA  LF          ; logische Filenummer
JSR  CLOSE       ; $FFC3, Datei schließen
```

### 6.3.3. Die Technik der Datenspeicherung - LOAD und SAVE

Zur Daten- und Programmspeicherung stehen Ihnen beim Commodore 64 zwei grundsätzliche Möglichkeiten zur Verfügung - Speicherung auf Kassette oder Diskette. Da sich die beiden Geräte in Möglichkeiten und Technik sehr unterscheiden, sollen sie getrennt beschrieben werden.

#### Datenspeicherung auf Kassette

Die Technik des Kassettenrekorders erlaubt es prinzipiell nur Daten sequentiell aufzuzeichnen und auch nur in der gleichen Reihenfolge wieder zu lesen. Ein direkter Zugriff auf bestimmte Daten ist also nicht möglich. Man muß solange lesen, bis man die gewünschten Daten erreicht hat. Ebenso ist ein Verändern von Daten nicht möglich. Man kann nur die Datei komplett lesen, die Änderung vornehmen und dann die Datei wieder komplett auf Band zurückschreiben.

Da zur Programmspeicherung nur ein sequentielles Schreiben und Lesen notwendig ist, bietet sich der Kassettenrekorder als preiswertes Medium zur Programmspeicherung an. Als Nachteil bleibt jedoch die geringe Geschwindigkeit, mit der dies geschieht. Dies ist ein prinzipieller Nachteil, da die Daten seriell (bitweise) übertragen und gespeichert werden. Aus Gründen der Datensicherheit werden die gesamten Daten zweimal hintereinander übertragen, um Aussetzer (drop outs) durch fehlerhafte Bandstellen korrigieren zu können.

Sehen wir uns jetzt die Technik der Datenspeicherung etwas genauer an.

Sollen Daten auf Band geschrieben werden, so muß das Band gestartet werden. Dies geschieht automatisch durch den Computer. Zuerst wird ein Ton zur Synchronisation auf das Band geschrieben und dann zweimal hintereinander die Daten. Damit diese zeitaufwendige Prozedur nicht zu oft geschehen muß, werden die zu schreibenden Daten zuerst in einem Puffer gesammelt bevor sie auf Band geschrieben werden. Der Bandpuffer ist 192 Zeichen lang und liegt im Commodore 64 von Adresse 828 bis 1019 (\$33C - \$3FB). Beim Einlesen werden die Daten aus dem Bandpuffer geholt. Da auf dem Band verschiedenartige Daten gespeichert werden sollen, muß eine Möglichkeit zur Unterscheidung geschaffen werden. Deshalb geht jeder Datenaufzeichnung ein Kopf 'Header' voraus, der die Informationen enthält. Der Header wird in den Puffer und dann auf Band geschrieben und später wieder von Band in den Puffer geladen.

Der Header ist folgendermaßen aufgebaut:

- |         |      |                           |
|---------|------|---------------------------|
| 1.      | Byte | Kennzeichen für Headertyp |
| 2.      | Byte | Startadresse, low Byte    |
| 3.      | Byte | Startadresse, high Byte   |
| 4.      | Byte | Endadresse, low Byte      |
| 5.      | Byte | Endadresse, high Byte     |
| 6.- 21. | Byte | Filenamen                 |

Der Headertyp (erstes Byte) hat folgende Bedeutung:

- |   |   |  |
|---|---|--|
| 1 | - | BASIC-Programm, wird ab BASIC-Start geladen  |
| 2 | - | Datenblock, Bytes 2 -192 enthalten die Daten |

- 3 - Maschinenprogramm, wird absolut geladen
- 4 - Datenheader, kündigt ein Datenfile an
- 5 - End of Tape Block, kennzeichnet Bandende

Handelt es sich um einen Programheader, Typ 1 oder 3, dann folgen in den den nächsten 4 Bytes die Programmstart- und Endadresse sowie anschließend der Programmname. Auf dem Band steht danach in einem Block das Programm (zweimal).

Ist der Headertyp 3, dann wird das Programm ab der Adresse geladen, die im Programheader steht. Beim Headertyp 1 wird die gelesene Startadresse verworfen und das Programm ab dem BASIC-Start geladen (normalerweise \$B00). Durch Angabe der Sekundäradresse 1 beim Laden läßt sich ein absolutes Laden (an die gelesene Startadresse) erzwingen. Dies ist bei Maschinenprogrammen unbedingt notwendig, da solche Programme nur in dem Speicherbereich laufen, für den sie auch geschrieben wurden.

Beim Headertyp 2 handelt es sich um Daten, die mit PRINT# auf Band geschrieben wurden. Bei jedem INPUT# oder GET# werden dann Daten aus dem Puffer gelesen. Ist der Puffer leer, wird die Programmausführung unterbrochen und der nächste Datenblock vom Band in den Puffer gelesen.

Wie kann man nun beim Abspeichern eines Programms entscheiden, ob es als BASIC-Programm oder als Maschinenprogramm gespeichert wird? Dies geschieht über die Sekundäradresse. Abspeichern ohne Sekundäradresse (oder 0) erzeugt ein BASIC-Programm, Headertyp 1. Sekundäradresse 1 (oder eine ungerade Zahl) erzeugt ein Maschinenprogramm. Sekundäradresse 2 oder 3 schreibt nach dem Programm noch einen End-Of-Tape Block.

Hier ist der Zusammenhang nochmal tabellarisch:

LADEN	
LOAD "NAME",1	- lädt BASIC-Programm verschieblich gekennzeichnetes Maschinenprogramm (Typ 3) wird absolut geladen
LOAD "NAME",1,1	- lädt jedes Programm absolut
SPEICHERN	
SAVE "NAME",1	- speichert als BASIC-Programm ab
SAVE "NAME",1,1	- speichert als Maschinenprogramm ab
SAVE "NAME",1,2	- speichert als BASIC-Programm mit zusätzlichem EOT-Block
SAVE "NAME",1,3	- speichert als Maschinenprogramm mit zusätzlichem EOT-Block

Beim Öffnen einer Banddatei hat die Sekundäradresse folgende Bedeutung:

OPEN 1,1,0, "NAME"	- öffnet Datei zum Lesen
OPEN 1,1,1, "NAME"	- öffnet Datei zum Schreiben
OPEN 1,1,2, "NAME"	- öffnet Datei zum Schreiben mit zusätzlichem EOT-Block

Wird beim Laden eines Programms oder beim Öffnen der Datei vom Band ein EOT-Block gefunden, so wird die Fehlermeldung 'file not found error' ausgegeben.

Der CLOSE-Befehl schließt eine Datei wieder. War die Datei zum Schreiben geöffnet, wird dadurch in den Bandpuffer ein Endekennzeichen geschrieben (Nullbyte) und der Puffer auf

Band geschrieben. Daraus wird klar, daß es unbedingt erforderlich ist, nach dem Schreiben auf eine Banddatei diese mit CLOSE zu schließen, da sonst die letzten Daten nicht auf Band geschrieben werden und verloren gehen. War die Sekundäradresse 2 angegeben, wird zusätzlich noch ein End-of-Tape Block (EOT) auf Band geschrieben.

Ebenso gibt es Einschränkungen bei der Datenübertragung auf Band. Normalerweise werden die Daten im ASCII-Format auf Band geschrieben (Buchstaben, Ziffern und Sonderzeichen). Werden mit PRINT#1, CHR\$(I) Daten geschrieben, so lassen sich nicht alle möglichen Zeichen schreiben. Insbesondere wird CHR\$(0) ausgefiltert, da es vom Betriebssystem als Endekennzeichen verwendet wird.

#### Datenspeicherung auf Diskette

Alle diese Einschränkungen bestehen bei der Datenübertragung auf Diskette nicht. Eine Diskette ist in einzelne Spuren und Sektoren unterteilt, auf die einzeln zugegriffen werden kann. Dadurch ist es möglich, direkt auf die gewünschten Daten zuzugreifen ohne erst eine Vielzahl anderer Daten überlesen zu müssen.

Programme werden folgendermaßen gespeichert:

Zuerst werden das low Byte und das high Byte der Programmadresse übertragen und unmittelbar danach das Programm selbst. Ein Unterschied zwischen BASIC-Programm und Maschinenprogramm wird beim Abspeichern nicht gemacht. Die Unterscheidung wird nur beim Laden gemacht.

LOAD "NAME",B	lädt BASIC-Programm Programmstartadresse wird ignoriert, es wird ab BASIC-Start geladen.
LOAD "NAME",B,1	lädt Maschinenprogramm, das Programm wird ab der gespeicherten Startadresse geladen.

Will man die Startadresse eines Programms auf Diskette wissen, kann man dies so machen:

```
10 OPEN 1,B,0,"NAME" :REM Programmdatei zu Lesen öffnen
20 GET#1, A$, B$ :REM Startadresse holen
30 IF A$ = "" THEN A$ = CHR$(0)
40 IF B$ = "" THEN B$ = CHR$(0)
50 PRINT ASC(A$)+256*ASC(B$):REM Adresse dezimal
60 CLOSE 1
```

Die Startadresse des Programms wird dezimal ausgedruckt. Die Abfrage auf den Leerstring in Zeile 30 und 40 ist deshalb erforderlich, weil der GET-Befehl ein Nullbyte nicht akzeptiert.

Will man zum Beispiel ein Maschinenprogramm auf Diskette schreiben, so kann dies so geschehen:

```
10 OPEN 1,B,1,"NAME":REM Programmdatei öffnen
20 PRINT#1, CHR$(AD-INT(AD/256)*256); :REM Adresse low
30 PRINT#1, CHR$(AD/256); :REM Startadresse high
40 FOR I=0 TO N-1: N Bytes speichern
50 PRINT#1, CHR$(PEEK(AD+I)); :NEXT
```

## 60 CLOSE 1 : REM Programmdatei schließen

Die Variable AD enthält dabei die Startadresse, N ist die Länge des Programms in Byte.

Wie kann man nun Programme oder beliebige Speicherbereiche von Maschinensprache aus abspeichern?

Auch hierzu existieren wieder zwei Betriebssystem-Routinen, die diese Arbeit übernehmen.

```
LOAD    $FFD5
SAVE    $FFD8
```

Die LOAD-Routine wird folgendermaßen benutzt:  
Der Akku wird mit Null geladen. Dies ist das Kennzeichen für LOAD. Wird die LOAD-Routine mit 1 im Akku aufgerufen, wird VERIFY durchgeführt. Die Sekundäradresse entscheidet, wohin geladen wird. Ist die Sekundäradresse ungleich null, so wird an die Adresse geladen, die im Programm gespeichert ist. Ist die Sekundäradresse gleich null, wird an die Adresse geladen, die im X- (low) und Y-Register (high) übergeben wird. Ferner müssen Geräteadresse und Filename gespeichert sein. Auch dazu gibt es ROM-Routinen.

Beispiel: Laden eines Maschinenprogramms von Diskette an die Originaladresse.

```
LDX    #8           ; Gerätenummer der Floppy
LDY    #1           ; Sekundäradresse
JSR    $FFBA        ; Fileparameter setzen
LDA    #5           ; Länge des Filenamens
LDX    #<NAME        ; Adresse des Filenamens
LDY    #>NAME        ; High Byte der Adresse
JSR    $FFBD        ; Parameter für Filenamens setzen
LDA    #0           ; LOAD-Flag
JSR    LOAD          ; Programm laden
STX    ADR           ; Endadresse low
STY    ADR+1        ; Endadresse high
```

Wie aus dem Beispiel ersichtlich, übergibt die LOAD-Routine im X- und Y-Register die Endadresse des geladenen Programms. Im nächsten Beispiel soll ein Programm vom Band ab Adresse \$6000 geladen werden, die gespeicherte Adresse wird ignoriert.

```
LDX    #2           ; Gerätenummer des Rekorders
LDY    #0           ; Sekundäradresse
JSR    $FFBA        ; Fileparameter setzen
LDA    #5           ; Länge des Filenamens
LDX    #<NAME        ; Adresse des Filenamens
LDY    #>NAME        ; High Byte der Adresse
JSR    $FFBD        ; Parameter für Filenamens setzen
LDA    #0           ; LOAD-Flag
LDX    #$00         ; Loadadresse low
LDY    #$60         ; Loadadresse high
JSR    LOAD          ; Programm laden
STX    ADR           ; Endadresse low
STY    ADR+1        ; Endadresse high
```

Mit dem Monitor für den Commodore 64 wird ein Programm immer absolut geladen und (auf Kassette) gespeichert.

Soll von einem BASIC-Programm aus ein Maschinenprogramm mit LOAD geladen werden, so ergeben sich einige Schwierigkeiten. Absolutes Laden läßt sich zwar leicht durch Angabe der Sekundaradresse 1 erreichen, es besteht jedoch noch ein zweites Problem. Nach jedem LOAD wird die Endadresse des Laden immer gleich der Endadresse des BASIC-Programms gesetzt und die Programmausführung beginnt wieder am Programmanfang. Dies ist für das Nachladen von BASIC-Programmen (Overlay) durchaus sinnvoll, macht jedoch beim Laden von Maschinenprogrammen oder sonstigen Speicherinhalten Probleme. In einem solchen Falle empfiehlt sich ein kleines Maschinenprogramm wie eins der obigen Beispiele, das z.B. vom BASIC-Programm aus mit SYS aufgerufen wird. Auch eine Parameterübergabe ist möglich, z.B. Filenamen und Geräteadresse. Aufruf und Anwendung der entsprechenden Routinen ist in Kapitel 6 beschrieben.

Abspeichern von Programmen oder beliebigen Speicherbereichen mit der SAVE-Routine geschieht ähnlich. Hierzu muß der SAVE-Routine die Start- und Endadresse mitgeteilt werden. Außerdem werden Geräteadresse sowie Länge und Adresse des Filenamens (beim Abspeichern auf Diskette unbedingt erforderlich) benötigt. Die Startadresse muß in der Zeropage an zwei aufeinander folgenden Adressen stehen (low und high Byte), im Akku wird ein Zeiger auf diese Adresse übergeben. Die Endadresse steht im X- (low Byte) und Y-Register (high Byte). Geräte- und Filenamenparameter werden wie bei der LOAD-Routine gesetzt. Wir wollen als Beispiel jetzt den Speicherbereich von \$C000 bis \$CFFF unter dem Namen "PROGRAMM" auf Diskette speichern und haben die Startadresse des Programms in \$FB/\$FC gespeichert.

```

LDX  #8           ; Gerätenummer der Floppy
JSR  $FFBA        ; Fileparameter setzen
LDA  #8           ; Länge des Filenamens
LDX  <NAME        ; Adresse des Filenamens
LDY  >NAME        ; High Byte der Adresse
JSR  $FFBD        ; Parameter für Filenamen setzen
LDA  $FB          ; Zeiger auf Startadresse
LDX  #00          ; Endadresse + 1 low
LDY  #D0          ; Endadresse + 1 high
JSR  SAVE         ; Programm speichern
NAME .ASC "PROGRAMM" ; Filename

```

Wie Sie aus dem Beispiel ersehen, wird der Inhalt der Endadresse nicht mehr abgespeichert, es muß deshalb immer die Endadresse plus eins angegeben werden. Zum Abschluß wollen wir noch ein BASIC-Programm ohne Filenamen mit EOT-Kennzeichen auf Kassette schreiben.

```

LDX  #1           ; Gerätenummer des Rekorders
LDY  #2           ; Sekundaradresse für End-of-Tape
JSR  $FFBA        ; Fileparameter setzen
LDA  #0           ; kein Filenamen
JSR  $FFBD        ; Parameter für Filenamen setzen
LDA  #$2B         ; Zeiger auf BASIC-Programmstart
LDX  #2D          ; Endadresse low des Programms
LDY  #2E          ; Endadresse high
JSR  SAVE         ; Programm speichern

```

#### 6.3.4. Die Programmierung der RS 232 Schnittstelle

RS 232 ist die Bezeichnung für eine Schnittstelle zur seriellen Datenübertragung. Auch die europäische Bezeichnung V 24 ist gebräuchlich. Was bedeutet nun serielle Übertragung und wann wird sie benutzt?

Bei der seriellen Übertragung werden nicht wie bei der parallelen Schnittstelle jeweils 8 Bits auf verschiedenen Leitungen gleichzeitig, sondern Bit für Bit nacheinander übertragen. Daraus ergeben sich schon Vor- und Nachteile der verschiedenen Übertragungsweisen. Die serielle Schnittstelle kommt mit weniger Leitungen aus, z.B. ist Datenübertragung über eine Telefonleitung möglich, dafür geht es jedoch nicht so schnell wie bei der parallelen Übertragung, während diese wiederum mehr Leitungen benötigt.

Das Betriebssystem des Commodore 64 enthält bereits die komplette Software zur Bedienung einer seriellen RS 232 Schnittstelle. Die Schnittstelle selbst ist als RS 232 Steckmodul erhältlich, das auf den ÜBER-Port gesetzt wird. Jetzt können Sie mit Ihrem Commodore 64 auch mit Geräten mit serieller Schnittstelle kommunizieren.

Das Betriebssystem hat der RS 232 Schnittstelle die Geräteadresse 2 zugeordnet. Wird ein logisches File mit Geräte Nummer 2 geöffnet, so legt das Betriebssystem zwei Puffer zu je 256 Byte als Ein- und Ausgabepuffer für die zu Übertragenden Daten an. Dieser Pufferbereich liegt normalerweise am Ende des BASIC-RAMs. Wird die RS 232 Schnittstelle in einem BASIC-Programm verwendet, sollte der OPEN-Befehl zuerst gegeben werden, da dabei alle Variablen gelöscht werden. Auch wird nicht geprüft, ob noch ausreichend Speicher vorhanden ist. Zu einer Zeit kann immer nur ein Datenkanal für RS 232 offen sein.

Die Parameter für die Datenübertragung werden durch ein Kontrollregister und ein Befehlsregister festgelegt. Diese beiden Register werden als die ersten beiden Zeichen des 'Filenames' übergeben.

Das Kontrollregister dient zur Definition der Baud-Rate sowie der Anzahl der zu Übertragenden Daten- und Stopbits. Die Baud-Rate bestimmt die Geschwindigkeit der Datenübertragung in Bits pro Sekunde, die Stopbits werden nach jedem Übertragenen Datenwort (5-8 Bits) gesandt.

Das Befehlsregister bestimmt Übertragungsart, Paritätsprüfung und Art des Handshake.

Beim Kontrollregister bestimmen die untersten 4 Bits die Baud-Rate nach folgender Tabelle:

Bit	3	2	1	0	dezimal	Baud-Rate
	0	0	0	0	0	Anwender (nicht implementiert)
	0	0	0	1	1	50
	0	0	1	0	2	75
	0	0	1	1	3	110
	0	1	0	0	4	134.5
	0	1	0	1	5	150

0	1	1	0	6	300	
0	1	1	1	7	600	
1	0	0	0	8	1200	
1	0	0	1	9	1800	
1	0	1	0	10	2400	
1	0	1	1	11	3600	(nicht implementiert)
1	1	0	0	12	4800	(nicht implementiert)
1	1	0	1	13	7200	(nicht implementiert)
1	1	1	0	14	9600	(nicht implementiert)
1	1	1	1	15	19200	(nicht implementiert)

Sie können also Baud-Raten zwischen 50 und 2400 programmieren. Die Anzahl der Datenbits wird durch Bit 5 und 6 bestimmt:

Bit 6	5	dezimal	Anzahl der Datenbits
0	0	0	8 Bits
0	1	32	7 Bits
1	0	64	6 Bits
1	1	96	5 Bits

Die Anzahl der Stopbits schließlich wird durch Bit 7 bestimmt:

Bit 7	dezimal	Anzahl der Stopbits
0	0	1 Stopbit
1	128	2 Stopbits

Das Befehlsregister ist folgendermaßen organisiert:

Bit 0	dezimal	Handshake
0	0	3-Draht Handshake
1	1	X-Draht Handshake
Bit 4	dezimal	Übertragungsart
0	0	Vollduplex
1	16	Halbduplex
Bit 7 6 5	dezimal	Paritätsprüfung
X X 0	0	keine Paritätsprüfung, kein 8. Datenbit
0 0 1	32	ungerade Parität
0 1 1	96	gerade Parität
1 0 1	160	keine Paritätsprüfung, 8. Datenbit immer 1
1 1 1	224	keine Paritätsprüfung, 8. Datenbit immer 0

Sie wollen einen RS 232 Datenkanal mit folgenden Parametern eröffnen:

```

Übertragungsrate 2400 Baud
7 Bit ASCII Daten
2 Stopbits
keine Paritätsprüfung
8. Datenbit immer 0
Vollduplex
3-Draht Handshake

```



Die OPEN-Anweisung sähe dann so aus:

OPEN 1, 2, 0, CHR\$(10+0+128)+CHR\$(0+0+224)

Das Statusregister beim Verkehr über die RS 232 Schnittstelle hat eine andere Bedeutung als in der normalen Datenübertragung. Es läßt sich in BASIC zwar auch über die Variable ST abfragen, wird jedoch bei jedem Lesen gelöscht. Soll der Statuswert daher für mehrere Abfragen benutzt werden, muß er erst einer anderen Variablen zugeordnet werden. ST gibt nur den RS 232 Status wieder, wenn der letzte Datenverkehr über RS 232 lief. Von einem Maschinenprogramm läßt sich der Status jedoch auch ohne Löschen lesen. Die Bedeutung der einzelnen Bits des RS 232 Status ist im folgenden beschrieben. Ein gesetztes Bit bedeutet dabei, daß die Bedingung aufgetreten ist.

Bit	Beschreibung
0	Paritätsfehler
1	Rahmenfehler
2	Empfängerpuffer voll
3	unbenutzt
4	CTS (Clear to send) Signal fehlt
5	unbenutzt
6	DSR (Data set ready) Signal fehlt
7	Break Signal empfangen

Geschieht die Programmierung der RS 232 Schnittstelle in Maschinensprache, so kann man die Ein- und Ausgabepuffer für die Datenübertragung in einen beliebigen Speicherbereich legen. Die Zeiger auf die Puffer werden beim OPEN-Befehl einmal gesetzt und können danach jedoch in einen anderen Speicherbereich gelegt werden. Die entsprechenden Zeiger liegen in der Zeropage und zwar zeigt \$F7/\$F8 auf den Eingabepuffer und \$F9/\$FA auf den Ausgabepuffer.

Die Programmierung der Ein- Ausgabe auf die RS 232 Schnittstelle geschieht genauso wie bei anderen Ausgabegeräten, als Geräteadresse wird 2 gewählt. Siehe dazu das Kapitel über Ein/Ausgabeprogrammierung.

Dazu noch einige Adressen für die RS 232 Ein/Ausgabe

\$0293	Kontrollwort
\$0294	Befehlsword
\$0298	Anzahl der Datenbits, wird bei OPEN berechnet
\$0297	RS 232 Statuswort

### 6.3.5. Der serielle IEC-Bus des Commodore 64

Der Commodore 64 hat zum Anschluß von Peripheriegeräten einen seriellen Bus, an dem mehrere Geräte gleichzeitig betrieben werden können. Der Bus ist analog dem IEC-Bus (IEEE 488) der großen CBM-Geräte konzipiert, die Daten werden jedoch nicht parallel (jeweils 8 Bit gleich ein Byte gleichzeitig) wie beim IEEE-488-Bus sondern seriell Bit für Bit übertragen. Dadurch ist die Übertragungsgeschwindigkeit kleiner.

Zuerst soll das Konzept des IEC-Bus kurz erläutert werden.

Da am IEC-Bus gleichzeitig mehrere Geräte betrieben werden können, müssen sie zu unterscheiden sein. Dazu dient die Gerätenummer oder Primaradresse. Der Commodore hat für den IEC-Bus die Gerätenummern 4 bis 15 reserviert. Soll nun ein Gerät angesprochen werden, so sendet der Bus-Controller, was in unserem Fall immer der Computer ist, ein 'Achtung'-Signal auf einer Steuerleitung, auch Attention genannt oder ATN abgekürzt. Danach sendet der Computer die Geräteadresse des Geräts, das angesprochen werden soll. Danach wird das Attention-Signal wieder zurückgesetzt. Als nächstes muß dem Gerät jetzt mitgeteilt werden, ob es Daten empfangen oder Daten senden soll. Der Vergleich von Zuhörer und Redner ist hier angebracht, im Englischen spricht man von Listener und Talker. Soll ein Gerät Daten empfangen, so sendet man einen LISTEN-Befehl. Werden Daten von dem Gerät erwartet, so schickt man einen TALK-Befehl. Danach kann noch eine Sekundäradresse gesandt werden, die eine bestimmte Betriebsart auswählt bzw. festlegt wie die Daten zu verarbeiten sind. Jetzt kann man dann entweder Daten schicken oder empfangen. Damit die Datenübertragung richtig funktioniert, muß sichergestellt werden, daß das nächste Byte erst dann übertragen wird, wenn der Empfänger das Byte auch erhalten und gegebenenfalls verarbeitet hat. Dazu ist das sogenannte Handshake da. Dazu teilt der Sender mit, wenn er Daten auf den Bus gelegt hat. Wenn der Empfänger die Daten erhalten hat und bereit ist, die nächsten Daten zu erhalten, teilt er dies dem Sender mit und die Übertragung kann weiter gehen. Dadurch geht die Übertragung immer mit der maximalen Geschwindigkeit vonstatten, die die beiden Geräte erlauben. Ist die Übertragung beendet, so wird das Gerät wieder deadressiert. Dazu sendet der Computer entweder UNTALK wenn das Gerät Daten gesandt hatte oder UNLISTEN sofern es Daten empfangen hatte. Jetzt ist der Bus wieder für die nächste Operation frei.

Wie läßt sich nun die Bedienung des IEC-Bus in Maschinensprache programmieren ?

Für alle Aufgaben stehen im Betriebssystem Unterprogramme zur Verfügung, die in der Sprungtabelle im obersten ROM-Bereich zusammengefaßt sind, siehe dazu die letzte Seite des ROM-Listings.

Als Beispiel wollen wir uns ansehen, wie wir die Fehlermeldung der Floppy-Disk einlesen können. Schauen wir erst, wie dies in BASIC gemacht wird.

```
10 OPEN 15,8,15 : REM öffnen des Fehlerkanals
20 INPUT#15,A$,B$,C$,D$ : REM Fehlermeldung holen
30 PRINT A$;"",B$;"",C$;"",D$ : REM und ausgeben
```

#### 40 CLOSE 15 : REM Kanal schließen

In BASIC ist dies wegen des INPUT-Befehls nur im Programmmodus möglich.  
Hier ist nun ein Maschinenprogramm, das die gleichen Dienste tut.

```

C000 A9 08      LDA  #8          ; Geräteadresse der Floppy
C002 B5 BA      STA  FA          ;
C004 20 B4 FF    JSR  TALK        ; Talk senden
C007 A9 6F      LDA  #15 + $60   ; Sekundäradr. 15 plus $60
C009 B5 B9      STA  SA          ;
C00B 20 96 FF    JSR  SECTALK     ; Sekundäradresse für Talk
C00E 20 A5 FF L JSR  IECIN       ; Zeichen von Floppy holen
C011 20 D2 FF    JSR  PRINT      ; auf Bildschirm ausgeben
C014 C9 0D      CMP  #13        ; ist es carriage return ?
C016 D0 F6      BNE  L          ; nein, weitere Zeichen
C018 20 AB FF    JSR  UNTALK     ; Untalk senden
C01A 60         RTS             ; fertig

```

Hier ein Ladeprogramm in BASIC:

```

100 FOR I = 49152 TO 49179
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 8,133,186, 32,180,255,169,111,133,185, 32
130 DATA 150,255, 32,165,255, 32,210,255,201, 13,208,246
140 DATA 32,171,255, 96
150 IF S <> 4169 THEN PRINT "FEHLER IN DATAS !!" : END
160 PRINT "OK !"

```

Von BASIC aus läßt das Programm sich mit SYS 12\*4096 aufrufen.

Mit einem etwas längeren Maschinenprogramm kann man auf einfache Weise auch das Inhaltsverzeichnis der Diskette anzeigen. Man erspart sich auf diese Weise das Directory mit LOAD "\$",8 als BASIC-Programm zu laden, wobei jedoch das jeweilige BASIC-Programm im Speicher verloren geht.

```

C000 A9 24      LDA  #" $"      ; Dollarzeichen als Filename
C002 B5 FB      STA  $FB        ; speichern
C004 A9 FB      LDA  #$FB       ; Adresse des Filenamens
C006 B5 BB      STA  FNADR
C008 A9 00      LDA  #0         ; high Byte
C00A B5 BC      STA  FNADR+1
C00C A9 01      LDA  #1         ; Länge des Filenamens
C00E B5 B7      STA  FNLEN
C010 A9 08      LDA  #8         ; Gerätenummer der Floppy
C012 B5 BA      STA  FA
C014 A9 60      LDA  #$60       ; Sekundäradresse für LOAD
C016 B5 B9      STA  SA
C018 20 D5 F3    JSR  SENDNAM    ; File mit Namen eröffnen
C01B A5 BA      LDA  FA
C01D 20 B4 FF    JSR  TALK      ; Talk senden
C020 A5 B9      LDA  SA
C022 20 96 FF    JSR  SECTALK   ; Sekundäradresse senden
C025 A9 00      LDA  #0
C027 B5 90      STA  STATUS     ; Status löschen
C029 A0 03      LDY  #3         ; ersten 3 Byte überlesen
C02B B4 FB      LI  STY  $FB     ; als Zähler merken
C02D 20 A5 FF    JSR  IECIN     ; Byte von Floppy holen
C030 B5 FC      STA  $FC        ; und merken

```

```

C032 A4 90      LDY STATUS      ; Status testen
C034 D0 2F      BNE L4
C036 20 A5 FF    JSR IECIN      ; Byte von Floppy holen
C039 A4 90      LDY STATUS      ; Status testen
C03B D0 28      BNE L4
C03D A4 FB      LDY $FB         ; Zähler holen
C03F 88         DEY             ; und erniedrigen
C040 D0 E9      BNE L1
C042 A6 FC      LDX $FC         ; Byte zuruckholen
C044 20 CD BD    JSR LNPRT      ; 16-Bit Zahl ausgeben
C047 A9 20      LDA #" "       ; Zahl der belegten Blocks
C049 20 D2 FF    JSR PRINT      ; Leerzeichen ausgeben
C04C 20 A5 FF L3 JSR IECIN      ; nachstes Byte holen
C04F A6 90      LDX STATUS      ; Status testen
C051 D0 12      BNE L4
C053 AA         TAX
C054 F0 06      BEQ L2         ; Null ? dann Zeilenende
C056 20 D2 FF    JSR PRINT      ; sonst ausgeben
C059 4C 4C C0    JMP L3        ; und nächstes Zeichen holen
C05C A9 0D      L2 LDA #13      ; carriage return
C05E 20 D2 FF    JSR PRINT      ; ausgeben
C061 A0 02      LDY #2         ; zwei Bytes fur Linkadresse
C063 D0 C6      BNE L1         ; weitermachen
C065 20 42 F6 L4 JSR CLSFIL     ; Datei schließen
C068 60         RTS

```

Hier wieder das Ladeprogramm:

```

100 FOR I = 49152 TO 49256
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 36,133,251,169,251,133,187,169, 0,133,188
130 DATA 169, 1,133,183,169, 8,133,186,169, 96,133,185
140 DATA 32,213,243,165,186, 32,180,255,165,185, 32,150
150 DATA 255,169, 0,133,144,160, 3,132,251, 32,165,255
160 DATA 133,252,164,144,208, 47, 32,165,255,164,144,208
170 DATA 40,164,251,136,208,233,166,252, 32,205,189,169
180 DATA 32, 32,210,255, 32,165,255,166,144,208, 18,170
190 DATA 240, 6, 32,210,255, 76, 76,192,169, 13, 32,210
200 DATA 255,160, 2,208,198, 32, 66,246, 96
210 IF S <> 15343 THEN PRINT "FEHLER IN DATAS !!" : END
220 PRINT "OK !"

```

Der Aufruf von BASIC geschieht wieder mit SYS 12\*4096. Es wird dann das Inhaltsverzeichnis der Diskette auf dem Bildschirm angezeigt, ohne daß ein gespeichertes BASIC-Programm verloren geht.

Statt des seriellen IEC-Bus kann es sich vor allem aus Geschwindigkeitsgründen lohnen, ein IEC-Bus Modul einzusetzen, das auf den Memory Expansion Port gesetzt wird. Sie können dann sämtliche Peripheriegeräte der großen Commodoregeräte benutzen, wie z.B. die großen Diskettendoppellaufwerke.

Bei der Programmierung des parallelen IEC-Bus ändern sich lediglich die Adressen der Routinen wie z.B. Ein- und Ausgabe eines Bytes auf den IEC-Bus oder Senden von TALK oder LISTEN. Programmieren Sie dagegen Ihre Ein- und Ausgabe auf den IEC-Bus über logische Dateien mit BASIN und BSOUT, (siehe Kapitel 6.3.) so sind überhaupt keine Änderungen erforderlich.

## 7.1 Die Belegung der Zero-Page und weiterer wichtiger Bereiche

hexadresse	Dezimal	Belegung
00	0	Datenrichtungsregister für Prozessorport
01	1	Prozessorport
02	2	unbenutzt
03 - 04	3 - 4	Vektor für Umwandlung Fließkomma nach Fest
05 - 06	5 - 6	Vektor Umwandlung Fest nach Fließkomma
07	7	Suchzeichen
08	8	Hochkomma-Flag
09	9	Speicher für Spalte beim TAB-Befehl
0A	10	Load = 0, Verify = 1, Flag des Interpreters
0B	11	Zeiger in Eingabepuffer, Anzahl der Dimensionen
0C	12	Flag für DIM
0D	13	Typflag \$00 = numerisch, \$FF = String
0E	14	Flag für Integer = \$80, Real = \$00
0F	15	Hochkomma-Flag bei LIST
10	16	Flag für FN
11	17	Flag für INPUT \$00, GET \$40, READ \$98
12	18	Vorzeichen bei ATN
13	19	aktives I/O-Gerat
14 - 15	20 - 21	Integer-Adresse, z.B. Zeilennummer
16	22	Zeiger auf Stringstack
17 - 18	23 - 24	Zeiger auf zuletzt verwendeten String
19 - 21	25 - 33	Stringstack
22 - 25	34 - 37	Zeiger für diverse Zwecke
26 - 2A	38 - 42	Register für Funktionsauswertung und Arithmetik
2B - 2C	43 - 44	Zeiger auf BASIC-Programmstart
2D - 2E	45 - 46	Zeiger auf Start der Variablen
2F - 30	47 - 48	Zeiger auf Start der Arrays
31 - 32	49 - 50	Zeiger auf Ende der Arrays
33 - 34	51 - 52	Zeiger auf Beginn der Strings
35 - 36	53 - 54	Hilfszeiger für Strings
37 - 38	55 - 56	Zeiger auf BASIC-RAM Ende
39 - 3A	57 - 58	augenblickliche BASIC-Zeilennummer
3B - 3C	59 - 60	vorherige BASIC-Zeilennummer
3D - 3E	61 - 62	Zeiger auf nächstes BASIC-Statement für CONT
3F - 40	63 - 64	augenblickliche Zeilennummer für DATA
41 - 42	65 - 66	Zeiger auf nächstes DATA-Element
43 - 44	67 - 68	Zeiger auf Herkunft der Eingabe
45 - 46	69 - 70	Variablenname
47 - 48	71 - 72	Variablenadresse
49 - 4A	73 - 74	Zeiger auf Variablenwert
4B - 4C	75 - 76	Zwischenspeicher für Programmzeiger
4D	77	Maske für Vergleichoperationen
4E - 4F	78 - 79	Zeiger für FN
50 - 53	80 - 83	Stringdescriptor
54	84	Konstante \$4C JMP für Funktionen
55 - 56	85 - 86	Sprungvektor für Funktionen
57 - 58	87 - 91	Register für Arithmetik, Akku#3
5C - 60	92 - 96	Register für Arithmetik, Akku#4
61 - 65	97 - 101	Fließkommaakku#1, FAC

Hexadresse	Dezimal	Belegung
66	102	Vorzeichen von FAC
67	103	Zähler für Polynomauswertung
68	104	Rundungsbyte für FAC
69 - 6D	105 - 109	Fließkommaakkumulator #2, ARG
6E	110	Vorzeichen von ARG
6F	111	Vergleichsbyte der Vorzeichen von FAC und ARG
70	112	Rundungsbyte für FAC
71 - 72	113 - 114	Zeiger für Polynomauswertung
73 - 7A	115 - 123	CHRGET - Routine, holt Zeichen aus BASIC-Text
7A - 7B	122 - 123	Programmzeiger
8B - 8F	139 - 143	letzter RND-Wert
90	144	Statuswort ST
91	145	Flag für Stop-Taste
92	146	Zeitkonstante für Band
93	147	Flag für LOAD \$00 oder VERIFY \$01
94	148	Flag bei IEC-Ausgabe
95	149	Ausgabepuffer für IEC-Bus
96	150	Flag für EDT vom Band empfangen
97	151	Zwischenspeicher für Register
98	152	Anzahl der offenen Files
99	153	aktives Eingabegerät
9A	154	aktives Ausgabegerät
9E	155	Parität für Band
9C	156	Flag für Byte empfangen
9D	157	Flag für Direkt-Modus \$B0, Programm \$00
9E	158	Band Pass 1 Checksumme
9F	159	Band Pass 2 Fehlerkorrektur
A0 - A2	160 - 162	Time
A3	163	Bitzähler für serielle Ausgabe
A4	164	Zähler für Band
A5	165	Zähler für Band schreiben
A6	166	Zeiger in Bandpuffer
A7 - AB	167 - 171	Arbeitsspeicher für Band ein/Ausgabe
AC - AD	172 - 173	Zeiger für Bandpuffer und Scrolling
AE - AF	174 - 175	Zeiger auf Programmende bei LOAD/SAVE
B0 - B1	176 - 177	Zeitkonstanten für Band-Timing
B2 - B3	178 - 179	Zeiger auf Bandpuffer
B4	180	Bitzähler für Band
B5	181	nächstes Bit für RS 232
B6	182	Puffer für auszugebendes Byte
B7	183	Länge des Filenamens
B8	184	logische Filenummer
B9	185	Sekundaradresse
BA	186	Gerätenummer
BB - BC	187 - 188	Zeiger auf Filenamen
BD	189	Arbeitsspeicher serielle Ein/Ausgabe
BE	190	Passzähler für Band
BF	191	Puffer für serielle Ausgabe
C0	192	Flag für Bandmotor
C1 - C2	193 - 194	Startadresse für Ein/Ausgabe vom Bildschirm
C3 - C4	195 - 196	Endadresse für Ein/Ausgabe vom Bildschirm
C5	197	Nummer der gedruckten Taste, 64 = keine Taste
C6	198	Anzahl der gedruckten Tasten
C7	199	Flag für RVS-Modus
C8	200	Zeilenende für Eingabe

Hexadresse	Dezimal	Belegung
C9	201	Cursorzeile für Eingabe
CA	202	Cursorspalte für Eingabe
CB	203	gedruckte Taste, keine Taste = 64
CC	204	Flag für Cursor 0=Cursor ein, 1=Cursor aus
CD	205	Zähler für Cursor blinken
CE	206	Zeichen unter dem Cursor
CF	207	Flag für Cursor 1= Ein-Phase, 0= Aus-Phase
D0	208	Flag für Eingabe von Tastatur oder Bildschirm
D1 - D2	209 - 210	Zeiger auf Start der aktuellen Bildschirmzeile
D3	211	Cursorspalte
D4	212	Flag für Hochkommaodus
D5	213	Länge der Bildschirmzeile
D6	214	Cursorzeile
D7	215	diverse Zwecke
D8	216	Anzahl der Inserts
D9 - F2	217 - 242	MSB der Bildschirmzeilenanfänge
F3 - F4	243 - 244	Zeiger in Farb-RAM
F5 - F6	245 - 246	Zeiger auf Tastatur-Dekodiertabelle
F7 - F8	247 - 248	Zeiger auf RS 232 Eingabepuffer
F9 - FA	249 - 250	Zeiger auf RS 232 Ausgabepuffer
*****		
00FF - 010A	255 - 266	Puffer für Umwandlung Fließkomma nach ASCII
0100 - 013E	256 - 318	Speicher für Korrektur bei Bandeingabe
0100 - 01FF	256 - 511	Prozessor Stack
0200 - 025B	512 - 600	BASIC Eingabepuffer
0259 - 0262	601 - 610	Tabelle der logischen Filenummern
0263 - 026C	611 - 620	Tabelle der Geratenummern
026D - 0276	621 - 630	Tabelle der Sekundaradresse
0277 - 0280	631 - 640	Tastaturpuffer
0281 - 0282	641 - 642	Start des BASIC-RAM
0283 - 0284	643 - 644	Ende des BASIC-RAM
0285	645	Timeout-Flag für seriellen IEC-Bus
0286	646	augenblickliche Farbe
0287	647	Farbe unter dem Cursor
0288	648	High-Byte Video-RAM
0289	649	Länge des Tastaturpuffers
028A	650	Flag für Repeatfunktion für alle Tasten
028B	651	Zähler für Repeat-Geschwindigkeit
028C	652	Zähler für Repeat-Verzögerung
028D	653	Flag für Shift, Commodore und CTRL (Bit 0,1 und 2)
028E	654	Shift-Flag
028F - 0290	655 - 656	Zeiger für Tastatur-Dekodierung
0291	657	Flag für Shift/Commodore gesperrt
0292	658	Flag für Scrollen
0293	659	RS 232 Kontrollwort
0294	660	RS 232 Befehlswort
0295 - 0296	661 - 662	Bit-Timing
0297	663	RS 232 Status
0298	664	Anzahl der Datenbits für RS 232
0299 - 029A	665 - 666	RS 232 Baud-Rate
029B	667	Zeiger auf empfangenes Byte RS 232
029C	668	Zeiger auf Input von RS 232
029D	669	Zeiger auf zu übertragendes Byte RS 232

Hexadresse	Decimal	Belegung
029E	670	Zeiger auf Ausgabe auf RS 232
029F - 02A0	671 - 672	Speicher für IRQ während Bandbetrieb
02A1	673	CIA 2 NMI-Flag
02A2	674	CIA 1 Timer A
02A3	675	CIA 1 Interruptflag
02A4	676	CIA 1 Flag für Timer A
02A5	677	Bildschirmzeile
02A6	678	Flag für PAL- (1) oder NTSC-Version (0)
02C0 - 02FE	704 - 766	Sprite 11
0300 - 0301	768 - 769	\$E38B Vektor für BASIC-Warmstart
0302 - 0303	770 - 771	\$A483 Vektor für Eingabe einer Zeile
0304 - 0305	772 - 773	\$A57C Vektor für Umwandlung in Interpretermode
0306 - 0307	774 - 775	\$A71A Vektor für Umwandlung in Klartext (LIST)
0308 - 0309	776 - 777	\$A7E4 Vektor für BASIC-Befehlsadresse holen
030A - 030B	778 - 779	\$AE86 Vektor für Ausdruck auswerten
030C	780	Akku für SYS-Befehl
030D	781	X-Reg für SYS-Befehl
030E	782	Y-Reg für SYS-Befehl
030F	783	Status-Register für SYS-Befehl
0310	784	\$4C JMP-Befehl für USR-Funktion
0311 - 0312	785 - 786	\$B248 USR-Vektor
0314 - 0315	788 - 789	\$EA31 IRQ-Vektor
0316 - 0317	790 - 791	\$FE66 BRK-Vektor
0318 - 0319	792 - 793	\$FE47 NMI-Vektor
031A - 031B	794 - 795	\$F34A OPEN-Vektor
031C - 031D	796 - 797	\$F291 CLOSE-Vektor
031E - 031F	798 - 799	\$F20E CHKIN-Vektor
0320 - 0321	800 - 801	\$F250 CKOUT-Vektor
0322 - 0323	802 - 803	\$F333 CLARCH-Vektor
0324 - 0325	804 - 805	\$F157 INPUT-Vektor
0326 - 0327	806 - 807	\$F1CA OUTPUT-Vektor
0328 - 0329	808 - 809	\$F6ED STOP-Vektor
032A - 032B	810 - 811	\$F13E GET-Vektor
032C - 032D	812 - 813	\$F32F CLALL-Vektor
032E - 032F	814 - 815	\$FE66 Warmstart-Vektor
0330 - 0331	816 - 817	\$F4A5 LOAD-Vektor
0332 - 0333	818 - 819	\$F5ED SAVE-Vektor
033C - 03FB	828 - 1019	Bandpuffer
0340 - 037E	832 - 894	Sprite 13
0380 - 03BE	896 - 958	Sprite 14
03C0 - 03FE	960 - 1022	Sprite 15



## 7.2 Die Adressen des BASIC-Routinen

Der BASIC-Interpreter des Commodore 64 ist mit dem des VC 20 identisch. Er ist lediglich in der Adresslage verschoben. Die Umrechnung einer Adresse des Commodore 64 in die entsprechende Adresse des VC 20 geschieht folgendermaßen: Bei Adressen von \$A000 bis \$BFFF wird einfach \$2000 dazuaddiert, aus \$AB60 wird die Adresse \$CB60 im VC 20. Bei Adressen von \$E000 bis \$E37A wird von der Commodore 64 Adresse 3 abgezogen. Aus \$E30E wird die VC 20 Adresse \$E30B.

Adresse	Beschreibung
A000	Startvektor
A002	NMI-Vektor
A004	'cbmbasic'
A00C	Adressen der BASIC-Befehle minus 1
A052	Adressen der BASIC-Funktionen
A080	Hierarchiekodes und Adressen der BASIC-Operatoren
A09E	Liste der BASIC-Befehlsworte
A19E	BASIC-Fehlermeldungen
A328	Adressen der Fehlermeldungen
A364	Meldungen des BASIC-Interpreters
A38A	Stapelsuchroutine für FOR-NEXT und GOSUB
A3BB	Blockverschieberoutine
A3FB	prüft auf Platz im Stapel
A40B	schafft Platz im Speicher
A435	Ausgabe von 'out of memory'
A437	Fehlermeldung ausgeben
A469	Break-Einsprung
A474	Ready-Einsprung
A480	Eingabe-Warteschleife
A49C	Löschen und Einfügen von Programmzeilen
A533	BASIC-Programmzeilen neu binden
A560	holt eine Zeile in den Eingabepuffer
A571	Ausgabe von 'string too long'
A579	Umwandlung einer Zeile in Interpreterkode
A613	Startadresse einer BASIC-Zeile suchen
A642	BASIC-Befehl NEW
A65E	BASIC-Befehl CLR
A68E	Programmzeiger auf BASIC-Start setzen
A69C	BASIC-Befehl LIST
A717	Interpreterkode in Befehlswort umwandeln
A742	BASIC-Befehl FOR
A7AE	Interpreterschleife, führt BASIC-Befehle aus
A7ED	führt einen BASIC-Befehl aus
A81D	BASIC-Befehl RESTORE
A82C	bricht Programm bei gedrückter Stop-Taste ab
A82F	BASIC-Befehl STOP
A831	BASIC-Befehl END
A857	BASIC-Befehl CONT
A871	BASIC-Befehl RUN
A883	BASIC-Befehl GOSUB
A8A0	BASIC-Befehl GOTO
A8D2	Basic-Befehl RETURN
A8FB	BASIC-Befehl DATA
A906	sucht nächstes Statement
A909	sucht nächste Zeile
A92B	BASIC-Befehl IF
A93B	BASIC-Befehl REM

## Adresse Beschreibung

A94B	BASIC-Befehl ON
A96B	sucht Adresse einer BASIC-Zeile
A9A5	BASIC-Befehl LET
AA80	BASIC-Befehl PRINT#
AA86	BASIC-Befehl CMD
AAA0	BASIC-Befehl PRINT
AB1E	String ausgeben
AB3E	Leerzeichen bzw. Cursor right ausgeben
AB4D	Fehlerbehandlung bei Eingabe
AB7B	BASIC-Befehl GET
ABA5	BASIC-Befehl INPUT#
ABBF	BASIC-Befehl INPUT
AC06	BASIC-Befehl READ
ACFC	'?extra ignored' und '?redo from start'
AD1D	BASIC-Befehl NEXT
ADB8	FRMNUM holt Ausdruck und prüft auf numerisch
ADBD	prüft auf numerisch
ADBF	prüft auf String
AD99	Ausgabe von 'typ mismatch'
AD9E	FRMEVL holt und wertet beliebigen Ausdruck aus
AE83	arithmetischen Ausdruck holen
AEAB	Fließkommakonstante Pi
AED4	BASIC-Befehl NOT
AEF1	holt Ausdruck in Klammern
AEF7	prüft auf 'Klammer zu'
AEFA	prüft auf 'Klammer auf'
AEFD	prüft auf 'Komma'
AEFF	prüft auf Zeichen im Akku
AF0B	Ausgabe von 'syntax error'
AF2B	holt Variable
AFE6	BASIC-Befehl OR
AFE9	BASIC-Befehl AND
B016	Vergleichsoperationen
B0B1	BASIC-Befehl DIM
B113	prüft auf Buchstabe
B194	berechnet Zeiger auf erstes Arrayelement
B1A5	Fließkommakonstante -32768
B1AA	FAC nach Integer wandeln
B245	Ausgabe von 'bad subscript'
B24B	Ausgabe von 'illegal quantity'
B34C	berechnet Arraygröße
B37D	BASIC-Funktion FRE
B39E	BASIC-Funktion POS
B3A6	Test auf Direkt-Modus
B3AB	Ausgabe von 'illegal direct'
B3AE	Ausgabe von 'undef'd function'
B3B3	BASIC-Befehl DEF
B3E1	FN-Syntax prüfen
B3F4	BASIC-Funktion FN
B465	BASIC-Funktion STR\$
B475	Stringverwaltung, Zeiger auf String berechnen
B4B7	String einrichten
B526	Garbage Collection, nichtgebrauchte Strings entfernen
B63D	Stringverknüpfung '+'
B6A3	Stringverwaltung FRESTR
B6EC	BASIC-Funktion CHR\$
B700	BASIC-Funktion LEFT\$
B72C	BASIC-Funktion RIGHT\$
B737	BASIC-Funktion MID\$

Adresse	Beschreibung
B77C	BASIC-Funktion LEN
B782	Stringparameter holen
B78B	BASIC-Funktion ASC
B79B	Holt Byte-Ausdruck (0 bis 255)
B7AD	BASIC-Funktion VAL
B7EB	Holt Adresse (0 bis 65535) und Byte-Wert (0 bis 255)
B7F7	FAC nach Adressformat wandeln (Bereich 0 bis 65535)
B80D	BASIC-Funktion PEEK
B824	BASIC-Befehl POKE
B82D	BASIC-Befehl WAIT
B849	$FAC = FAC + 0.5$
B850	Minus $FAC = \text{Konstante (A/Y)} - FAC$
B853	Minus $FAC = ARG - FAC$
B867	Plus $FAC = \text{Konstante (A/Y)} + FAC$
B86A	Plus $FAC = ARG + FAC$
B97E	Ausgabe von 'overflow'
B9BC	Fließkommakonstanten für LOG
B9EA	BASIC-Funktion LOG
BA28	Multiplikation $FAC = \text{Konstante (A/Y)} * FAC$
BA2B	Multiplikation $FAC = ARG * FAC$
BA8C	$ARG = \text{Konstante (A/Y)}$
BAE2	$FAC = FAC * 10$
BAF9	Fließkommakonstante 10
BAFE	$FAC = FAC / 10$
BB0F	$FAC = \text{Konstante (A/Y)} / FAC$
BB12	$FAC = ARG / FAC$
BBBA	Ausgabe von 'division by zero'
BBA2	$FAC = \text{Konstante (A/Y)}$
BBC4	Akku#4 = FAC
BBCA	Akku#3 = FAC
BBD0	Variable = FAC
BBFC	$FAC = ARG$
BC0C	$ARG = FAC$
BC1B	FAC runden
BC2B	Vorzeichen von FAC holen
BC39	BASIC-Funktion SGN
BC5B	BASIC-Funktion ABS
BC5B	Konstante (A/Y) mit FAC vergleichen
BC9B	Umwandlung FAC nach Integer
BCCC	BASIC-Funktion INT
BCF3	Umwandlung ASCII nach Fließkomma
BDB3	Fließkommakonstanten für Fließkomma nach ASCII
BDC2	Ausgabe der Zeilennummer bei Fehlermeldung
BDCD	Positive Integerzahl (0 bis 65535) ausgeben
BDDD	FAC nach ASCII-Format wandeln
BF11	Fließkommakonstante 0.5
BF16	Binärzahlen für Umwandlung FAC nach ASCII
BF71	BASIC-Funktion SQR
BF7B	Potenzierung $FAC = \text{Konstante (A/Y)} \text{ hoch } FAC$
BF7B	Potenzierung $FAC = ARG \text{ hoch } FAC$
BFBF	Fließkommakonstanten für EXP
BFED	BASIC-Funktion EXP
E043	Polynomrechnung
E059	Polynomrechnung
E0BD	Fließkommakonstanten für RND
E097	BASIC-Funktion RND
E107	Ausgabe von 'break'
E10C	BSOUT ein Zeichen ausgeben
E112	BASIN ein Zeichen empfangen

# Adresse      Beschreibung

E118	CKOUT Ausgabegerät festsetzen
E11E	CHKIN Eingabegerät festsetzen
E124	GETIN ein Zeichen holen
E12A	BASIC-Befehl SYS
E156	BASIC-Befehl SAVE
E165	BASIC-Befehl VERIFY
E168	BASIC-Befehl LOAD
E1BE	BASIC-Befehl OPEN
E1C7	BASIC-Befehl CLOSE
E1D4	Parameter für LOAD und SAVE holen
E219	Parameter für OPEN holen
E264	BASIC-Funktion COS
E26B	BASIC-Funktion SIN
E2B4	BASIC-Funktion TAN
E2E0	Fließkommakonstanten für SIN und COS
E30E	BASIC-Funktion ATN
E33E	Fließkommakonstanten für ATN
E37B	BASIC-NMI-Einsprung
E394	BASIC-Kaltstart
E3A2	Kopie der CHRGET-Routine
E3BA	Anfangswert für RND-Funktion
E3BF	RAM für BASIC initialisieren
E447	Tabelle der BASIC-Vektoren
E453	BASIC-Vektoren laden

### 7.3 Vergleichstabelle VC 20 - Commodore 64

64	VC 20	Beschreibung
E45F	E429	Meldungen des Betriebssystems
E4E0	-	wartet auf Commodore-Taste
E4EC	-	Konstanten für RS 232 Timing
E500	E500	holt BASIC-Adresse des CIAs bzw. VIAs
E505	E505	holt Bildschirmformat Zeilen/Spalten
E50A	E50A	Cursor setzen bzw. Cursorposition holen
E518	E518	Bildschirm-Reset
E544	E55F	Bildschirm löschen
E566	E581	Cursor Home
E5A0	E58B	Videocontroller initialisieren
E5B4	E5CF	Zeichen aus Tastaturpuffer holen
E5CA	E5E5	Warteschleife für Tastatureingabe
E632	E64F	ein Zeichen vom Bildschirm holen
E6B4	E6B8	testet auf Hochkomma
E6B6	E6EA	MSB für Zeilenanfänge berechnen
EBDA	E921	Tabelle der Farbkodes
EBEA	E975	Bildschirm scrollen
E9C8	EA56	Zeile nach oben schieben
E9FF	EA8D	Bildschirmzeile löschen
EA1C	EEA1	Zeichen und Farbe auf Bildschirm setzen
EA24	EA82	Zeiger auf Farb-RAM berechnen
EA31	EABF	Interrupt-Routine
EAB7	EB1E	Tastaturabfrage
EB48	EBDC	Prüfung auf Shift, CTRL und Commodore-Taste
EB79	EC46	Zeiger auf Tastatur-Dekodiertabellen
EB81	EC5E	Dekodiertabellen
EC44	ED21	Prüfung auf Steuerzeichen
EC78	ED69	Dekodiertabellen
ECB9	EDE4	Konstanten für Videocontroller
ECE7	EDF3	'load (cr) run (cr)'
ECF0	EDFE	Tabelle der LSB der Bildschirmanfänge
ED09	EE14	TALK senden
ED0C	EE17	LISTEN senden
ED40	EEE4	ein Byte auf IEC-Bus ausgeben
EDB9	EEC0	Sekundäradresse für LISTEN senden
EDC7	EECE	Sekundäradresse für TALK senden
EDEF	EEF6	UNTALK senden
EDFE	EF04	UNLISTEN senden
EE13	EF19	ein Byte vom IEC-Bus holen
EEB3	EF96	Verzögerung eine Millisekunde
EEBB	EFA3	RS 232 Ausgabe
EF4A	F027	Anzahl der RS 232 Datenbits berechnen
F014	F0ED	Ausgabe in RS 232 Puffer
F0B6	F14F	GET von RS 232
F0A4	F160	Timer für IEC-Timeout setzen
F0BD	F174	Fehlermeldungen des Betriebssystems
F12B	F1E0	Meldungen ausgeben
F157	F20E	BASIN ein Zeichen holen
F1CA	F27A	BSOUT ein Zeichen ausgeben
F20E	F2C7	CHKIN festlegen des Eingabegeräts
F250	F309	CKOUT festlegen des Ausgabegeräts
F291	F34A	CLOSE
F30F	F3CF	logische Filenummer suchen
F31F	F3DF	Fileparameter setzen
F32F	F3EF	CLALL schließt alle I/O-Kanäle

64	VC 20	Beschreibung
F333	F3F3	CLRCH schließt I/O-Kanal
F34A	F40A	OPEN
F49E	F542	LOAD
F5AF	F647	'searching for filename' ausgeben
F5D2	F66A	'loading/verifying' ausgeben
F5DD	F675	SAVE
F68F	F728	'saving filename' ausgeben
F69B	F734	UDTIM laufende Zeit erhöhen
F6DD	F760	Time holen
F6E4	F767	Time setzen
F6ED	F770	Stop-Taste abfragen
F6FB	F77E	Fehlermeldungen des Betriebssystems ausgeben
F72C	F7AF	Programmheader vom Band lesen
F76A	F7E7	Header auf Band schreiben
F7D0	F84D	Startadresse des Bandpuffers holen
F7D7	F854	Start und Endadresse des Bandpuffers setzen
F7EA	F867	Bandheader nach Namen suchen
F80D	F88A	Bandpufferzeiger erhöhen
F817	F894	wartet auf Bandtaste für lesen
F82E	F8AB	fragt Bandtaste ab
F838	F8B7	wartet auf Bandtaste für schreiben
F841	F8C0	Block vom Band lesen
F84A	F8C9	Programm vom Band laden
F864	F8EA	Bandpuffer auf Band schreiben
F86B	F8EA	Block bzw. Programm auf Band schreiben
F8BE	F92F	I/O-Abschluß abwarten
F8E1	F94B	testet auf Stop-Taste
F92C	F9BE	Interrupt-Routine für Band lesen
FB97	F8DB	Bitzähler für serielle Ausgabe setzen
FBA6	F8EA	ein Bit auf Band schreiben
FBCD	FC0B	Interrupt-Routine für Band schreiben
FCB8	FCF6	IRQ-Vektor setzen
FCCA	FD0B	Bandmotor ausschalten
FCD1	FD11	prüft auf Erreichen der Endadresse
FCD8	FD1B	Adresszeiger erhöhen
FCE2	FD22	RESET
FD02	FD3F	prüft auf ROM in \$B000 bzw. \$A000
FD10	FD4D	ROM-Modul Identifizierung
FD15	FD52	Hardware und I/O Vektoren setzen bzw. holen
FD30	FD6D	Tabelle der Hardware und I/O-Vektoren
FD50	FD8D	Arbeitsspeicher initialisieren
FD9B	FD6D	Tabelle der IRQ-Vektoren
FDF9	FE49	Parameter für Filenamen setzen
FE00	FE50	Parameter für aktives File setzen
FE07	FE57	Status holen
FE18	FE66	Flag für Meldungen des Betriebssystems setzen
FE1C	FE6A	Status setzen
FE21	FEF6	Timeout-Flag für IEC-Bus setzen
FE25	FE73	RAM-Obergrenze setzen bzw. holen
FE34	FE82	RAM-Untergrenze setzen bzw. holen
FE43	FEA9	NMI-Routine
FEC2	FF5C	Konstanten für RS 232 Baud-Rate
FF48	FF72	Interrupthandler
FFB1	FFBA	Sprungtabelle der Betriebssystem-Routinen

## 7.4 Vergleichstabelle CBM 8000 - Commodore 64

Um die Übertragung von Maschinenprogrammen, die für die Commodore-Rechner der 8000er Serie geschrieben wurden, auf den Commodore 64 zu erleichtern, wurden die entsprechenden Adressen des BASIC-Interpreters von CBM 8000 und C 64 gegenübergestellt.

8000	64	Bedeutung
B000	A00C	Adressen der BASIC-Befehle (minus 1)
B066	A052	Adressen der BASIC-Funktionen
B094	A080	Hierarchiekodes und Adressen der BASIC-Operatoren
B0B2	A09E	Liste der BASIC-Befehlswoorte
B20D	A19E	BASIC-Fehlermeldungen
B322	A38A	Stapelsuchroutine für FOR-NEXT und GOSUB
B350	A38B	Blockverschieberoutine
B393	A3FB	prüft auf Platz im Stack
B3CD	A435	Ausgabe von 'out of memory'
B3CF	A437	Fehlermeldung ausgeben
B3FF	A474	Ready-Modus
B406	A480	Eingabe-Warteschleife
B4B6	A533	BASIC-Programmzeilen neu binden
B4E2	A560	holt eine Zeile in den Eingabepuffer
B4FB	A579	Umwandlung einer Zeile in Interpreterkode
B5A3	A613	sucht eine BASIC-Zeile
B5D2	A642	BASIC-Befehl NEW
B5EE	A65E	BASIC-Befehl CLR
B622	A68E	Programmzeiger auf BASIC-Start setzen
B630	A69C	BASIC-Befehl LIST
B6B5	A717	Interpreterkode in Befehlswoort umwandeln
B6DE	A742	BASIC-Befehl FOR
B74A	A7AE	Interpreterschleife, führt BASIC-Befehle aus
B7B5	A7ED	führt einen BASIC-Befehl aus
B7B7	AB1D	BASIC-Befehl RESTORE
B7C6	AB2C	STOP und END
B7EE	AB57	BASIC-Befehl CONT
B80B	AB71	BASIC-Befehl RUN
B813	AB83	BASIC-Befehl GOSUB
B830	ABA0	BASIC-Befehl GOTO
B85D	ABD2	BASIC-Befehl RETURN
B8B3	ABFB	BASIC-Befehl DATA
B891	A906	sucht nächstes Statement
B894	A909	sucht nächste Zeile
B8B3	A92B	BASIC-Befehl IF
B8C6	A93B	BASIC-Befehl REM
B8D6	A94B	BASIC-Befehl ON
B8F6	A96B	sucht Adresse einer BASIC-Zeile
B930	A9A5	BASIC-Befehl LET
BA0B	AA80	BASIC-Befehl PRINT#
BABE	AA86	BASIC-Befehl CMD
BAA8	AAA0	BASIC-Befehl PRINT
BB1D	AB1E	String ausgeben
BB4C	AB4D	Fehlerbehandlung bei Eingabe
BB7A	AB7B	BASIC-Befehl GET
BBA4	ABA5	BASIC-Befehl INPUT#
BBBE	ABBF	BASIC-Befehl READ
BCF7	ACFC	'?extra ignored' und '?redo from start'
BD19	AD1D	BASIC-Befehl NEXT
BD84	AD8A	FRMNUM holt Ausdruck und prüft auf numerisch
BDB7	ADB0	prüft auf numerisch

B000	64	Bedeutung
BD89	ADB8	prüft auf String
BD93	AD99	Ausgabe von 'type mismatch'
BD98	AD9E	FRMEVL holt und wertet beliebigen Ausdruck aus
BDD1	AE83	nächsten Element eines Ausdrucks holen
BEA0	AEAB	Fließkommakonstante Pi
BEE9	AEF1	holt arithmetischen Ausdruck in Klammern
BEEF	AEF7	prüft auf 'Klammer zu'
BEF2	AEFA	prüft auf 'Klammer auf'
BEF5	AEFD	prüft auf 'Komma'
BEF7	AEFF	prüft auf Zeichen im Akku
BF00	AF08	Ausgabe von 'syntax error'
BF0C	AF28	holt Variable
C086	AFE6	BASIC-Operator OR
C089	AFE9	BASIC-Operator AND
C086	B016	Vergleichsoperationen
C121	B081	BASIC-Befehl DIM
C186	B113	prüft auf Buchstabe
C2D9	B1A5	Fließkommakonstante -32768
C2DD	B1AA	FAC nach Integer wandeln
C370	B245	Ausgabe von 'bad subscript'
C373	B248	Ausgabe von 'illegal quantity'
C477	B34C	berechnet Arraygröße
C4A8	B37D	BASIC-Funktion FRE
C4C9	B39E	BASIC-Funktion POS
C4CF	B3A6	Test auf Direkt-Modus
C4DC	B3B3	BASIC-Befehl DEF
C50A	B3E1	FN-Syntax prüfen
C51D	B3F4	BASIC-Funktion FN
C58E	B465	BASIC-Funktion STR\$
C59E	B475	Stringverwaltung, Zeiger auf String berechnen
C5B0	B487	String einrichten
C66A	B526	Garbage collection
C74F	B63D	Stringverknüpfung
C7B5	B6A3	Stringverwaltung, FRESTR
C822	B6EC	BASIC-Funktion CHR\$
C836	B700	BASIC-Funktion LEFT\$
C862	B72C	BASIC-Funktion RIGHT\$
C86D	B737	BASIC-Funktion MID\$
C8B2	B77C	BASIC-Funktion LEN
C8B8	B782	Stringparameter holen
C8C1	B78B	BASIC-Funktion ASC
C8D1	B78B	holt Byte-Ausdruck (0 bis 255)
C8E3	B7AD	BASIC-Funktion VAL
C921	B7EB	holt Adresse und Byte-Wert
C92D	B7F7	FAC nach Adressformat wandeln (Bereich 0 bis 65535)
C943	B80D	BASIC-Funktion PEEK
C95A	B824	BASIC-Befehl POKE
C963	B82D	BASIC-Befehl WAIT
C97F	B849	FAC = FAC + 0.5
C986	B850	Minus FAC = Konstante (A/Y) - FAC
C989	B853	Minus FAC = ARG - FAC
C99D	B867	Plus FAC = Konstante (A/Y) + FAC
C9A0	B86A	Plus FAC = ARG + FAC
CAB4	B97E	Ausgabe von 'overflow'
CAF2	B9BC	Fließkommakonstanten für LOG
CB20	B9EA	BASIC-Funktion LOG
CB5E	BA28	Multiplikation FAC = Konstante (A/Y) * FAC
CB61	BA2B	Multiplikation FAC = ARG * FAC
CBC2	BABC	ARG = Konstante (A/Y)



B000	64	Bedeutung
CC1B	BAE2	$FAC = FAC * 10$
CC2F	BAF9	Fließkommakonstante 10
CC34	BAFE	$FAC = FAC / 10$
CC45	BB0F	Division $FAC = \text{Konstante } (A/Y) / FAC$
CC4A	BB12	Division $FAC = ARG / FAC$
CCCO	BBBA	Ausgabe von 'division by zero'
CCD8	BBA2	$FAC = \text{Konstante } (A/Y)$
CCFD	BBC4	Akku#4 = FAC
CD03	BBCA	Akku#3 = FAC
CD0A	BBD0	Variable = FAC
CD32	BBFC	$FAC = ARG$
CD42	BC0C	$ARG = FAC$
CD51	BC1B	FAC runden
CD61	BC2B	Vorzeichen von FAC holen
CD6F	BC39	BASIC-Funktion SGN
CD8E	BC5B	BASIC-Funktion ABS
CD91	BC5B	Konstante (A/Y) mit FAC vergleichen
CDD1	BC9B	FAC nach Integer wandeln
CE02	BCCC	BASIC-Funktion INT
CE29	BCF3	Umwandlung ASCII nach FAC
CEE9	BDB3	Fließkommakonstanten für Fließkomma nach ASCII
CF7B	BDC2	Ausgabe der Zeilennummer bei Fehlermeldung
CF93	BDD0	FAC nach ASCII-Format wandeln
DOC2	BF11	Fließkommakonstante 0.5
DOC7	BF16	Binärzahlen für Umwandlung FAC nach ASCII
D10B	BF71	BASIC-Funktion SQR
D112	BF7B	Potenzierung $FAC = \text{Konstante } (A/Y) \text{ hoch } FAC$
D115	BF7B	Potenzierung $FAC = ARG \text{ hoch } FAC$
D156	BFBF	Fließkommakonstanten für EXP
D1B4	BFE0	BASIC-Funktion EXP
D1D7	E043	Polynomberechnung
D1ED	E059	Polynomberechnung
D221	E0BD	Fließkommakonstanten für RND
D229	E097	BASIC-Funktion RND
D2B2	E264	BASIC-Funktion COS
D2B9	E26B	BASIC-Funktion SIN
D2D2	E2B4	BASIC-Funktion TAN
D2FE	E2E0	Fließkommakonstanten für SIN und COS
D32C	E30E	BASIC-Funktion ATN
D35C	E33E	Fließkommakonstanten für ATN
D399	E3A2	Kopie der CHRGET-Routine

## 7.5 Umsetzung von VC-20 Programmen auf Commodore 64

Der Commodore 64 ist als der große Bruder des VC-20 schon in seinem äußeren erkennbar. Allerdings steckt in ihm einiges mehr, als der "Kleine" zu bieten hat. Angafangen bei der hochauflösenden Farbgraphik, über die Sprites, bis hin zur Möglichkeit der Tonerzeugung mittels Synthesizer - alles das unterscheidet ihn vom VC-20. Trotzdem möchte man natürlich nicht auf das Angebot der VC-20 Software verzichten.

Leider ist es nicht möglich, die Module oder Kassetten des VC-20 einfach zu übernehmen. Es gibt aber genügend Programmlistings, die man sicherlich gerne auf den Commodore 64 übertragen möchte.

Man muß hier zwischen zwei verschiedenen Programmarten unterscheiden:

### 1.) Die Maschinenprogramme

Bei diesen Programmen ist die Adaption zwar nicht sehr schwierig, aber trotzdem komplizierter als bei einfachen BASIC Programmen. Um diese Programme adaptieren zu können, müssen Sie die Vergleichstabelle (Kapitel 7.1) mit zu Rate ziehen. Treffen Sie zum Beispiel an eine Sprungadresse, die auf eine Adresse im ROM zeigt, heißt es, diese Adresse in der Vergleichstabelle des Commodore 64 zu finden, um dann diese in dem Programm gegen die Adresse des VC-20 auszutauschen. Generell läßt sich sagen, daß man den Unterschied zwischen den einzelnen Adressen auch selber berechnen kann.

So ist zum Beispiel die Adresse bei VC-20, die die Anfangsadresse der BASIC GET-Routine ist, Hex CB7B. Beim Commodore 64 liegt sie um genau Hex 2000 niedriger, also ab Adresse \$AB7B. Ab E000 muß beim Commodore 64 der Wert 3 subtrahiert werden, um die Adresse beim VC-20 zu erhalten.

### 2.) Die BASIC Programme

Bei BASIC Programmen sind die Änderungen nicht so kompliziert. An erster Stelle ist hier natürlich das andere Bildschirmformat zu nennen. Hier müssen Sie darauf achten, daß Sie entweder das Bildschirmformat der VC-20 Programme völlig neu gestalten, oder aber nach den 22 Zeichen ein RETURN einfügen, damit die anderen Zeichen nicht in die selbe Zeile rutschen.

Die einzige Schwierigkeit bei den BASIC Programmen bilden die POKE Befehle. Hier muß man zwischen 2 Arten unterscheiden.

#### 1.) POKE Befehle in den Bildschirmspeicher

Hier müssen Sie die ganzen Adressen ändern. Vergleichen Sie hierzu die Tabellen der Zeichengeneratoren und der Bildschirmseite.

#### 2.) Werte die das ROM des VC-20 betreffen

Dazu müssen Sie genau so vorgehen, wie wir es eben bei der Änderung der Maschinenprogramme besprochen haben.

## 7.6 Umsetzung von CBM Programmen auf den Commodore 64

Vielen Leuten ist es überhaupt noch nicht aufgefallen, daß es eine sehr große Ähnlichkeit zwischen dem Commodore 64 und seinen großen Kollegen gibt. Es ist so natürlich auch möglich, Programme von diesen Rechnern, wie zum Beispiel vom schon legendären PET, zu übernehmen. Bei den meisten Geräten haben wir sogar das selbe Bildschirmformat. Natürlich kommen wir auch hier nicht ohne gewisse Änderungen aus. Doch haben alle Commodore dieser Klasse einen Vorteil: Ihre BASIC Version ist voll übertragbar. Das heißt, daß Sie keine Änderungen in BASIC Programmen machen müssen, sofern es keine POKE Befehle sind.

Für diese Befehle, müssen Sie die entsprechenden Adressen bei den Commodore Rechnern kennen, die sich vom Commodore 64 unterscheiden. Es gibt bekanntlich eine große Auswahl an Literatur über die PET/CBM Geräte. In unserem Literaturverzeichnis finden Sie die entsprechenden Bücher aufgeführt.

Generell ist zu sagen: BASIC und Maschinensprache unterscheiden sich bei diesen Geräten nicht. Der 6510 ist voll mit dem 6502 kompatibel. Die einzigen Änderungen beziehen sich daher auf die unterschiedlichen Einsprungsadressen zu den verschiedenen Routinen im Betriebssystem bzw. die unterschiedlichen Adressen der Bildschirmspeicher und die Vektoren der Zeropage.

Wenn Sie dieses beachten, können Sie mit etwas Übung ohne größere Schwierigkeiten alle Programme der PET/CBM/VC-20 Geräte auf dem Commodore 64 anwenden. Und denken Sie daran: Durch eine Erweiterung dieser Programme, können Sie sogar Farbe, Ton und Graphik des Commodore 64 ausnutzen - und welcher PET/CBM hat das schon zu bieten? Außerdem steht Ihnen beim Commodore 64 ein mit 64KB RAM ausreichend großer Arbeitsspeicher zur Verfügung, so daß die Programm nach den Änderungen auf dem Commodore 64, besser sein können als vorher.

Noch ein Tip am Rande: VC-20 Kassetten lassen sich auf dem Commodore 64 dann lesen, wenn diese zuerst auf einen CBM Rechner eingeladen und danach wieder direkt auf Kassette abgespeichert wurden.

Sie haben aber auch die Möglichkeit auf dem Commodore 64 einen CBM zu simulieren. Diese sogenannte Emulation läßt sich mit folgendem Programm verwirklichen:

```
10 POKE 56576,6: REM CBM-BILDSCHIRM NACH 32768
20 POKE 53272,4
30 POKE 648,128: REM BASIC ERFAHRT BILDSCHIRMAADRESSE
40 POKE 1024,0: POKE 44,4: POKE 56,128: REM BASIC START/ENDE
50 PRINT CHR$(147)
60 NEW
```

Viel Spaß bei der Emulation.

### 8.1 Die Nutzung des Commodore 64 ROM-Listing

Auf den folgenden über 100 Seiten finden Sie nun das Assemblerlisting des gesamten Betriebssystems und des BASIC-Interpreters Ihres Commodore 64. Wollen Sie tiefer in die Arbeitsweise Ihres Rechners eindringen, so können Sie sich durch ein Studium dieses Listings viele Anregungen für die eigene Assemblerprogrammierung holen. Den größten Nutzen werden Sie jedoch daraus ziehen können, wenn Sie die Routinen des Betriebssystems und des BASIC-Interpreters in Ihren eigenen Programmen als Unterprogramme verwenden. Sie können sich dadurch viel Programmierarbeit ersparen. Wie findet man nun am schnellsten die benötigten Routinen? Die Adressen aller BASIC-Befehle stehen zu Beginn des BASIC-Interpreters als Sprungtabelle auf die eigentlichen Routinen. Wollen Sie sich z.B. den GOTO-Befehl näher ansehen, so finden Sie in der Sprungtabelle die Adresse \$ABAO. Dort steht dann der eigentliche Befehl. Wollen Sie arithmetische Aufgaben im Fließkommaformat durchführen (so wie die normalen Variablen gespeichert werden), so finden Sie die benötigten Routinen ab Adresse \$BB00 und folgende. Sehen Sie dazu auch Kapitel 5. Das Betriebssystem beginnt ab Adresse \$E400. Zuerst steht dort der Bildschirmeditor, anschließend Tastaturbedienung und Interruptroutine. Es folgen dann Routinen zur Ein- Ausgabe für den seriellen IEC-Bus, RS 232 Schnittstelle sowie die Bedienung des Kassettengeräts. Auch hier sind die wichtigsten Routinen wieder als Sprungtabelle zusammengefaßt, diesmal am Ende des ROMs ab Adresse \$FFB1. Im Kapitel 7 sind die Einsprungpunkte noch einmal aufgeführt.

Am Beispiel der Blockverschieberoutine des BASIC-Interpreters soll einmal gezeigt werden, wie man solche Routinen nutzt.

Oft steht man bei der Maschinenprogrammierung vor der Aufgabe, einen Speicherbereich zu verschieben. Das kann z.B. ein Grafikbild oder eine Tabelle sein. Anstatt die Verschiebung nun selber zu programmieren und auszutesten, kann man auf eine bestehende Routine zurückgreifen. Die ganze Aufgabe reduziert sich jetzt auf die Parameterübergabe und den Aufruf des Unterprogramms. Die Einsprungadresse ist \$A3BF. Wollen wir z.B. den Bereich von \$24BA bis \$29C0 einschließlich so verschieben, daß er bei \$3000 endet. Dazu müssen die Adressen des alten Blockanfangs und des Endes sowie die Adresse des neuen Blockendes übergeben werden. In unserem Falle sähe das so aus:

```
LDA  $$BA
LDY  $$24      ; Zeiger auf alten Blockanfang
STA  $5F
STY  $60
LDA  $$C1
LDY  $$29      ; Zeiger auf altes Blockende + 1
STA  $5A
STY  $5B
```

```

LDA  #$01
LDY  #$30    ; Zeiger auf neues Blockende + 1
STA  $58
STY  $59
JSR  $A3BF    ; Aufruf der Verschieberoutine

```

Mit dieser Routine lassen sich beliebige Speicherbereiche zu höheren Adressen verschieben. Der BASIC-Interpreter benötigt diese Routine, um den kompletten Arraybereich nach oben zu schieben, wenn eine neue Variable angelegt werden muß.

Wer selbst auf dem 64er in Maschinensprache programmieren will, dem wird das Studium des ROM-Listings nicht nur die Arbeitsweise von BASIC-Interpreter und Betriebssystem näher bringen. Durch geschickte Nutzung der Routinen läßt sich nicht nur viel Programmierzeit, sondern auch Speicherplatz sparen, der dann für Ihre eigenen Anwendungen zur Verfügung steht.

Auf den nächsten Seiten finden Sie nun zum leichteren Durchfinden ein alphabetisches Verzeichnis sämtlicher ROM-Routinen und ihrer Adressen im Listing.

## 8.2 Alphabetisches Verzeichnis der ROM-Routinen

Abfrage auf gedrückte Bandtaste	\$F82E
Adresse eines Arrayelements berechnen	\$B30E
Adressen der Basic-Befehle (minus 1)	\$A00C
Adressen der Basic-Funktionen	\$A052
Adressen der Fehlermeldungen	\$A32B
Adresszeiger erhöhen	\$FCDB
Anfangswert für RND-Funktion	\$E3BA
Arbeitsspeicher initialisieren	\$FD50
Arrayelement suchen	\$B2E9
Arrayvariable anlegen	\$B261
Ausgabe der Zeilennummer bei Fehlermeldung	\$BDC2
Ausgabe eines Fragezeichens	\$AB45
Ausgabe eines Leerzeichens	\$AB3B
Ausgabe in RS 232 Puffer	\$F014
ARG = Konstante (A/Y)	\$BABC
ARG nach FAC übertragen	\$BBFC
ASCII-Kode nach Bildschirakode wandeln	\$E691
Band für Lesen vorbereiten	\$F8E2
Bandheader nach Namen suchen	\$F7EA
Bandpuffer auf Band schreiben	\$F864
Bandpufferzeiger erhöhen	\$F80D
Basic CKOUT-Routine	\$E4AD
Basic Kaltstart	\$E394
Basic NMI-Einsprung	\$E37B
Basic-Befehl CLOSE	\$E1C7
Basic-Befehl CLR	\$A65E
Basic-Befehl CMD	\$AA86
Basic-Befehl CONT	\$A857
Basic-Befehl DATA	\$ABFB
Basic-Befehl DEF	\$B3B3
Basic-Befehl DIM	\$B0B1
Basic-Befehl END	\$AB31
Basic-Befehl FOR	\$A742
Basic-Befehl GET	\$AB7B
Basic-Befehl GOSUB	\$AB83
Basic-Befehl GOTO	\$ABA0
Basic-Befehl IF	\$A92B
Basic-Befehl INPUT	\$ABBF
Basic-Befehl INPUT#	\$ABA5
Basic-Befehl LET	\$A9A5
Basic-Befehl LIST	\$A69C
Basic-Befehl LOAD	\$E16B
Basic-Befehl NEW	\$A642
Basic-Befehl NEXT	\$AD1D
Basic-Befehl ON	\$A94B
Basic-Befehl ON	\$A94B
Basic-Befehl OPEN	\$E1BE
Basic-Befehl POKE	\$BB24
Basic-Befehl PRINT	\$AAA0
Basic-Befehl PRINT#	\$AAB0
Basic-Befehl READ	\$AC06
Basic-Befehl REM	\$A93B
Basic-Befehl RESTORE	\$AB1D
Basic-Befehl RETURN	\$ABD2
Basic-Befehl RUN	\$AB71
Basic-Befehl SAVE	\$E156
Basic-Befehl STOP	\$AB2F

Basic-Befehl SYS	\$E12A
Basic-Befehl VERIFY	\$E165
Basic-Befehl WAIT	\$B82D
Basic-Befehlsworte	\$A09E
Basic-Fehlermeldungen	\$A19E
Basic-Funktion ABS	\$BC58
Basic-Funktion ASC	\$B78B
Basic-Funktion ATN	\$E30E
Basic-Funktion CHR\$	\$B6EC
Basic-Funktion COS	\$E264
Basic-Funktion EXP	\$BFED
Basic-Funktion FN	\$B3F4
Basic-Funktion FRE	\$B37D
Basic-Funktion INT	\$8CCC
Basic-Funktion LEFT\$	\$B700
Basic-Funktion LEN	\$B77C
Basic-Funktion LOG	\$B9EA
Basic-Funktion MID\$	\$B737
Basic-Funktion PEEK	\$B80D
Basic-Funktion POS	\$B39E
Basic-Funktion RIGHT\$	\$B72C
Basic-Funktion RND	\$E097
Basic-Funktion SGN	\$BC39
Basic-Funktion SIN	\$E26B
Basic-Funktion SQR	\$BF71
Basic-Funktion STR\$	\$B465
Basic-Funktion TAN	\$E2B4
Basic-Funktion VAL	\$B7AD
Basic-Kode in Klartext wandeln	\$A717
Basic-Operator AND	\$AFE9
Basic-Operator NOT	\$AE04
Basic-Operator OR	\$AFE6
Basic-Routine BASIN	\$E112
Basic-Routine BSOUT	\$E10C
Basic-Routine CHKIN	\$E11E
Basic-Routine CKOUT	\$E11B
Basic-Routine GETIN	\$E124
Basic-Statement ausführen	\$A7ED
Basic-Vektoren laden	\$E453
Basisadresse des CIAs holen	\$E500
Betriebssystem-Meldungen	\$E45F
Bildschirm löschen	\$E544
Bildschirm scrollen	\$E8EA
Bildschirm-Reset	\$E51B
Bildschirmformat holen	\$E505
Bildschirmzeile löschen	\$E9FF
Bit auf Band schreiben	\$FBA6
Bitweise Multiplikation	\$BA59
Bitzähler für serielle Ausgabe setzen	\$FB97
Block vom Band lesen	\$FB41
Blockverschiebe-Routine	\$A3BB
Byte auf seriellen Bus ausgeben	\$ED40
Byte auf seriellen Bus ausgeben	\$EDDD
Byte vom seriellen Bus holen	\$EE13
Byte vom Band holen	\$F199
Byte von RS 232 holen	\$F1BB
Bytewert nach X holen, GETBYT	\$B79B
BASIN-Routine	\$F157
BSOUT-Routine	\$F1CA
Cursor setzen/holen	\$E50A

Cursor Home	\$E566
Cursorposition berechnen	\$E56C
CHKIN-Routine	\$F20E
CKOUT-Routine	\$F250
CLALL-Routine	\$F32F
CLOSE-Routine	\$F291
CLRCH-Routine	\$F333
Datenbits für RS 232 berechnen	\$EF4A
Dimensionierte Variable holen	\$B1D1
Division FAC = ARG / FAC	\$BB12
Division FAC = Konstante (A/Y) / FAC	\$BB0F
Einfügen einer Fortsetzungszeile	\$E963
Eingabe einer Zeile	\$A560
Eingabe-Warteschleife	\$A480
Fehlerbehandlung bei Eingabe	\$AB4D
Fehlermeldung ausgeben	\$A437
Fehlermeldung des Betriebssystems	\$F6FB
Fileparameter setzen	\$F31F
Flag für Systemmeldungen setzen	\$FE18
Fließkommakonstante -32768	\$B1A5
Fließkommakonstante 0.5	\$BF11
Fließkommakonstante 10	\$BAF9
FAC = FAC * 10	\$BAE2
FAC = FAC + 0.5	\$BB49
FAC = FAC / 10	\$BAFE
FAC = Konstante (A/Y)	\$BBAA
FAC nach Akku#3 übertragen	\$BBCA
FAC nach Akku#4 übertragen	\$BBC7
FAC nach ARG übertragen	\$BC0C
FAC nach ASCII wandeln und nach \$100	\$BDD0
FAC nach Variable übertragen	\$BBD0
FN-Syntax prüfen	\$B3E1
FRESTR	\$B6A3
FRMEVL Auswerten eines beliebigen Ausdrucks	\$AD9E
FRMNUM Ausdruck holen und auf numersich prüfen	\$AD8A
Garbage Collection	\$B526
GETADR und GETBYT, 16- und 8-Bit-Wert holen	\$B7EB
GETADR, FAC in positive 16-Bit-Zahl wandeln	\$B7F7
GETIN-Routine	\$F13E
Hardware und I/O-Vektoren setzen/holen	\$FD15
Header auf Band schreiben	\$F76A
Hierarchiekodes der Basic-Operatoren	\$A080
Hilfsroutine für Arrayberechnung	\$B34C
Hintergrundfarbe setzen	\$E4DA
Interpreterschleife	\$A7AE
Interruptroutine	\$EA31
Interruptroutine für Band lesen	\$F92C
Interruptroutine für Band schreiben	\$FBCD, \$FC6A
IRQ-Einsprung	\$FF48
IRQ-Vektor setzen	\$FC88
IRQ-Vektoren	\$FD98
Kernal Sprungtabelle	\$FF81
Konstante P1	\$AEAD
Konstanten für ATN	\$E33E
Konstanten für EXP	\$BFBF
Konstanten für Fließkomma nach ASCII	\$BF16
Konstanten für Fließkomma nach ASCII	\$BDB3
Konstanten für LOG	\$B98C
Konstanten für RND	\$E08D
Konstanten für SIN und COS	\$E2E0



Konstanten für Umwandlung TI nach TI\$	\$BF3A
Kopie der CHRGET-Routine	\$E3A2
Listen senden	\$ED0C
Logische Filenummer suchen	\$F30F
Löschen und Einfügen von Programmzeilen	\$A49C
LOAD-Routine	\$F49E
Mantisse von FAC invertieren	\$B947
Meldungen des Betriebssystems	\$F0BD
Meldungen des Betriebssystems ausgeben	\$F12B
Meldungen des Interpreters	\$A364
Minus FAC = ARG - FAC	\$B853
Minus FAC = Konstante (A/Y) - FAC	\$B850
Multiplikation FAC = ARG * FAC	\$BA2B
Multiplikation FAC = Konstante (A/Y) * FAC	\$BA2B
MSB für Zeilenanfänge neu berechnen	\$E6B6
Nächste Zeile suchen	\$A909
Nächstes Element eines Ausdrucks holen	\$AE83
Nächstes Statement suchen	\$A906
NMI-Einsprung	\$FE43
NMI-Routine für RS 232	\$FED6
Obergrenze RAM setzen/holen	\$FE25
OPEN-Routine	\$F34A
Parameter für aktives File setzen	\$FE00
Parameter für Filenamen setzen	\$FDF9
Parameter für LOAD und SAVE holen	\$E1D4
Parameter für OPEN holen	\$E219
Platz für String reservieren	\$B4F4
Plus FAC = ARG + FAC	\$B86A
Plus FAC = Konstante (A/Y) + FAC	\$B867
Polynomberechnung 1	\$E043
Polynomberechnung 2	\$E039
Positive Integerzahl in A/X ausgeben	\$BDCD
Potenzierung FAC = ARG hoch FAC	\$BF7B
Potenzierung FAC = ARG hoch Konstante (A/Y)	\$BF7B
Programm vom Band laden	\$F84A
Programmheader vom Band lesen	\$F72C
Programmzeiger auf Basic-Start	\$A6BE
Programmzeile einfügen	\$A4ED
Programmzeile löschen	\$A4A9
Programmzeilen neu binden	\$A533
Prüfung auf numerisch	\$AD8D
Prüfung auf Auto-Start-ROM	\$FD02
Prüfung auf Buchstabe	\$B113
Prüfung auf Erreichen der Endadresse	\$FCD1
Prüfung auf Klammer auf	\$AEFA
Prüfung auf Klammer zu	\$AEF7
Prüfung auf Komma	\$AEFD
Prüfung auf Platz im Speicher	\$A3F8
Prüfung auf Shift, CTRL, Commodore	\$E848
Prüfung auf Steuerzeichen	\$EC44
Prüfung auf Stop-Taste	\$AB2C
Prüfung auf String	\$AD8F
Prüfung auf Systemvariable	\$AF14
Prüfung auf Übereinstimmung mit laufendem Zeichen	\$AEFD
Rechtsverschieben eines Registers	\$B983
Rekordermotor ausschalten	\$FCCA
Reset-Routine	\$FCE2
RAM für Basic initialisieren	\$E3BF
ROM-Modul Identifizierung	\$FD10
RS 232 Ausgabe	\$EEBB

RS 232 Ausgabe	\$F20B
RS 232 CHKIN	\$F04D
RS 232 GET	\$F0B6
Schafft Platz im Speicher	\$A40B
Sekundäradresse nach Listen senden	\$EDB9
Sekundäradresse nach Talk senden	\$EDC7
Stapelsuch-Routine	\$A3BE
Startadresse des Bandpuffers holen	\$F7D0
Startadresse einer Programmzeile berechnen	\$A613
Status holen	\$FE07
Stoptaste abfragen	\$F6ED
String ausgeben	\$AB1E
String holen, Zeiger nach A/Y	\$B4B7
String in reservierten Bereich übertragen	\$B67A
Stringparameter holen	\$B7B2
Stringparameter vom Stack holen	\$B761
Stringvergleich	\$B02E
Stringverknüpfung	\$B63D
Stringverwaltung, FRESTR	\$B6A3
Stringzeiger berechnen	\$B475
SAVE-Routine	\$F5DD
Tabelle der Basic-Vektoren	\$E447
Tabelle der Farbkodes	\$E8DA
Tabelle der Hardware- und I/O-Vektoren	\$FD30
Tabelle der LSB der Bildschirmzeilen-Anfänge	\$ECF0
Talk senden	\$ED09
Tastatur Dekodiertabelle 1	\$EBB1
Tastatur Dekodiertabelle 2	\$EBC2
Tastatur Dekodiertabelle 3	\$EC03
Tastatur Dekodiertabelle 4	\$EC7B
Tastaturabfrage	\$EAB7
Term in Klammern holen	\$AEF1
Test auf Direkt-Modus	\$B3A6
Test auf Hochkomma	\$E6B4
Test auf Stop-Taste	\$FBD0
Time erhöhen	\$F69B
Time holen	\$F6DD
Time setzen	\$F6E4
Timeout-Flag für seriellen Bus setzen	\$FE21
Timerkonstanten für RS 232 Baud Rate, NTSC-Version	\$FEC2
Timerkonstanten für RS 232 Baud Rate, PAL-Version	\$E4EC
Umwandlung einer Zeile in Interpreterkode	\$A579
Umwandlung ASCII nach Fließkommaformat	\$BCF3
Umwandlung Fließkomma nach Integer	\$B1B2
Umwandlung Fließkomma nach Integer	\$BC9B
Unlisten senden	\$EDFE
Untalk senden	\$EDEF
Untergrenze RAM setzen/holen	\$FE34
Variable anlegen	\$B11D
Variable holen	\$AF2B
Variable holen	\$B0BB
Vergleich	\$B016
Vergleich Konstante (A/Y) mit FAC	\$BC5B
Videocontroller initialisieren	\$E5A0
Vorzeichen von FAC holen	\$BC2B
Warten auf Bandtaste	\$FB17
Warten auf Bandtaste für Schreiben	\$FB3B
Warten auf Commodore-Taste	\$E4E0
Warteschleife für Tastatureingabe	\$E5CA
Wertzuweisung an normalen String	\$AA2C

Wertzuweisung INTEGER	\$A9C4
Wertzuweisung REAL	\$A9D6
Wertzuweisung String	\$A9D9
Zeichen auf Bildschirm ausgeben	\$E716
Zeichen auf Ziffer prüfen	\$AA1D
Zeichen aus Tastaturpuffer holen	\$E5B4
Zeichen und Farbe auf Bildschirm setzen	\$EA1C
Zeichen vom Bildschirm holen	\$E632
Zeiger auf erstes Arrayelement berechnen	\$B194
Zeiger auf Farbram berechnen	\$EA24
Zeiger auf Tastaturdekodiertabellen	\$EB79
Zeile nach oben schieben	\$E9CB
Zeilennummer holen und in Adressformat wandeln	\$A96B
Zeit holen	\$AFB4

### B.3 Commodore 64 ROM-Listing

\*\*\*\*\*

A000 94 E3                      Start-Vektor    \$E394  
A002 7B E3                      NMI-Vektor     \$E37B

\*\*\*\*\*

A004 43 42 4D 42 41 53 49 43    'cbmbasic'

\*\*\*\*\*                      Adressen der BASIC-Befehle (minus 1)

	Interpretercode	Adresse	Befehl
A00C 30 AB	\$80	\$AB31	END
A00E 41 A7	\$81	\$A742	FOR
A010 1D AD	\$82	\$AD1E	NEXT
A012 F7 AB	\$83	\$ABF8	DATA
A014 A4 AB	\$84	\$AB45	INPUT
A016 BE AB	\$85	\$ABBF	INPUT
A018 B0 B0	\$86	\$B081	DIM
A01A 05 AC	\$87	\$AC06	READ
A01C A4 A9	\$88	\$A9A5	LET
A01F 9F AB	\$89	\$ABA0	GOTO
A020 70 AB	\$8A	\$AB71	RUN
A022 27 A9	\$8B	\$A928	IF
A024 1C AB	\$8C	\$AB1D	RESTORE
A026 B2 AB	\$8D	\$AB83	GOSUB
A028 D1 AB	\$8E	\$ABD2	RETURN
A02A 3A A9	\$8F	\$A93B	REM
A02C 2E AB	\$90	\$AB2F	STOP
A02F 4A A9	\$91	\$A948	ON
A030 2C BB	\$92	\$B82D	WAIT
A032 67 E1	\$93	\$E168	LOAD
A034 55 E1	\$94	\$E156	SAVE
A036 64 E1	\$95	\$E165	VERIFY
A038 B2 B3	\$96	\$B3B3	DEF
A03A 23 BB	\$97	\$B824	POKE
A03C 7F AA	\$98	\$AA80	PRINT
A03E 9F AA	\$99	\$AAA0	PRINT
A040 56 AB	\$9A	\$AB57	CONT
A042 9B A6	\$9B	\$A69C	LIST
A044 5D A6	\$9C	\$A65E	CLR
A046 B5 AA	\$9D	\$AA86	CMD
A048 29 E1	\$9E	\$E12A	SYS
A04A BD E1	\$9F	\$E1BE	OPEN
A04C C6 E1	\$A0	\$E1C7	CLOSE
A04E 7A AB	\$A1	\$AB78	GET
A050 41 A6	\$A2	\$A642	NEW

\*\*\*\*\*

\*\*\*\*\*                      Adressen der BASIC-Funktionen

A052 39 BC	\$B4	\$BC39	SGN
A054 CC BC	\$B5	\$BCCC	INT
A056 58 BC	\$B6	\$BC58	ABS
A058 10 03	\$B7	\$0310	USR
A05A 7D B3	\$B8	\$B37D	FRE
A05C 9E B3	\$B9	\$B39E	FOS
A05E 71 BF	\$BA	\$BF71	SQR
A060 97 E0	\$BB	\$E097	RND
A062 EA B9	\$BC	\$B9EA	LOG
A064 ED BF	\$BD	\$BFED	EXP
A066 64 E2	\$BE	\$E264	COS

A06B 6B E2	\$BF	\$E26B	SIN
A06A BA E2	\$C0	\$E2B4	TAN
A06C 0E E3	\$C1	\$E30E	ATN
A06E 0D BB	\$C2	\$B80D	PEEK
A070 7C B7	\$C3	\$B77C	LEN
A072 65 B4	\$C4	\$B465	STR\$
A074 AD B7	\$C5	\$B7AD	VAL
A076 BB B7	\$C6	\$B7BB	ASC
A07B EC B6	\$C7	\$B6EC	CHR\$
A07A 00 B7	\$C8	\$B700	LEFT\$
A07C 2C B7	\$C9	\$B72C	RIGHT\$
A07E 37 B7	\$CA	\$B737	MID\$

\*\*\*\*\* Hierarchiecodes und Adressen-1 der Operatoren

A080 79 69 BB	\$79, \$B86A	Addition
A083 79 52 BB	\$79, \$B853	Subtraktion
A086 7B 2A BA	\$7B, \$BA2B	Multiplikation
A089 7B 11 BB	\$7B, \$B812	Division
A08C 7F 7A BF	\$7F, \$BF7B	Potenzierung
A08F 50 E8 AF	\$50, \$AFE9	AND
A092 46 E5 AF	\$46, \$AFE6	OR
A095 7D B3 BF	\$7D, \$BFB4	Vorzeichenwechsel
A09B 5A D3 AE	\$5A, \$AED4	NOT
A09B 64 15 B0	\$64, \$B016	Vergleich

\*\*\*\*\* BASIC-Befehlsworte

A09E 45 4E	end
A0A0 C4 46 4F D2 4E 45 5B D4	for next
A0AB 44 41 54 C1 49 4E 50 55	data input#
A0B0 54 A3 49 4E 50 55 D4 44	input dia
A0BB 49 CD 52 45 41 C4 4C 45	read let
A0C0 D4 47 4F 54 CF 52 55 CE	goto run
A0CB 49 C6 52 45 53 54 4F 52	if restore
A0D0 C5 47 4F 53 55 C2 52 45	gosub return
A0DB 54 55 52 CE 52 45 CD 53	rem stop
A0E0 54 4F D0 4F CE 57 41 49	on wait
A0EB D4 4C 4F 41 C4 53 41 56	load save
A0F0 C5 56 45 52 49 46 D9 44	verify def
A0FB 45 C6 50 4F 4B C3 50 52	poke print#
A100 49 4E 54 A3 50 52 49 4E	print
A10B D4 43 4F 4E D4 4C 49 53	cont list
A110 D4 43 4C D2 43 4D C4 53	clr cmd sys
A11B 59 D3 4F 50 45 CE 43 4C	open close
A120 4F 53 C5 47 45 D4 4E 45	get new
A12B D7 54 41 42 AB 54 CF 46	tab# to
A130 CE 53 50 43 AB 54 4B 45	spcl then
A13B CE 4E 4F D4 53 54 45 D0	not stop
A140 AB AD AA AF DE 41 4E C4	* - * / ^ and
A14B 4F D2 BE BD BC 53 47 CE	or (<=>) sgn
A150 49 4E D4 41 42 D3 55 53	int abs usr
A15B D2 46 52 C5 50 4F D3 53	fre pos sqr
A160 51 D2 52 4E C4 4C 4F C7	rnd log
A16B 45 5B D0 43 4F D3 53 49	exp cos sin
A170 CE 54 41 CE 41 54 CE 50	tan atn peek
A17B 45 45 CB 4C 45 CE 53 54	len str\$
A180 52 AA 56 41 CC 41 53 C3	val asc
A18B 43 4B 52 AA 4C 45 46 54	chr\$ left\$
A190 AA 52 49 47 4B 54 AA 4D	right\$ mid\$
A19B 49 44 AA 47 CF 00	go

\*\*\*\*\*

A19E 54 4F  
A1A0 4F 20 4D 41 4E 59 20 46  
A1AB 49 4C 45 D3 46 49 4C 45  
A1B0 20 4F 50 45 CE 46 49 4C  
A1B8 45 20 4E 4F 54 20 4F 50  
A1C0 45 CE 46 49 4C 45 20 4E  
A1CB 4F 54 20 46 4F 55 4E C4  
A1D0 44 45 56 49 43 45 20 4E  
A1DB 4F 54 20 50 52 45 53 45  
A1E0 4E D4 4E 4F 54 20 49 4E  
A1EB 50 55 54 20 46 49 4C C5  
A1F0 4E 4F 54 20 4F 55 54 50  
A1FB 55 54 20 46 49 4C C5 4D  
A200 49 53 53 49 4E 47 20 46  
A208 49 4C 45 20 4E 41 4D C5  
A210 49 4C 4C 45 47 41 4C 20  
A218 44 45 56 49 43 45 20 4E  
A220 55 4D 42 45 D2 4E 45 58  
A22B 54 20 57 49 54 46 4F 55  
A230 54 20 46 4F D2 53 59 4E  
A23B 54 41 DB 52 45 54 55 52  
A240 4E 20 57 49 54 48 4F 55  
A248 54 20 47 4F 53 55 C2 4F  
A250 55 54 20 4F 46 20 44 41  
A25B 54 C1 49 4C 4C 45 47 41  
A260 4C 20 51 55 41 4E 54 49  
A26B 54 D9 4F 56 45 52 46 4C  
A270 4F D7 4F 55 54 20 4F 46  
A278 20 4D 45 4D 4F 52 D9 55  
A280 4E 44 45 46 27 44 20 53  
A28B 54 41 54 45 4D 45 4E D4  
A290 42 41 44 20 53 55 42 53  
A29B 43 52 49 50 D4 52 45 44  
A2A0 49 4D 27 44 20 41 52 52  
A2AB 41 D9 44 49 56 49 53 49  
A2B0 4F 4E 20 42 59 20 5A 45  
A2BB 52 CF 49 4C 4C 45 47 41  
A2C0 4C 20 44 49 52 45 43 D4  
A2CB 54 59 50 45 20 4D 49 53  
A2D0 4D 41 54 43 CB 53 54 52  
A2DB 49 4E 47 20 54 4F 4F 20  
A2E0 4C 4F 4E C7 46 49 4C 45  
A2EB 20 44 41 54 C1 46 4F 52  
A2F0 4D 55 4C 41 20 54 4F 4F  
A2FB 20 43 4F 4D 50 4C 45 DB  
A300 43 41 4E 27 54 20 43 4F  
A30B 4E 54 49 4E 55 C5 55 4E  
A310 44 45 46 27 44 20 46 55  
A31B 4E 43 54 49 4F CE 56 45  
A320 52 49 46 D9 4C 4F 41 C4

# BASIC-Fehlermeldungen

- 1 too many files
- 2 file open
- 3 file not open
- 4 file not found
- 5 device not present
- 6 not input file
- 7 not output file
- 8 missing filename
- 9 illegal device number
- 10 next without for
- 11 syntax
- 12 return without gosub
- 13 out of data
- 14 illegal quantity
- 15 overflow
- 16 out of memory
- 17 undef'd statement
- 18 bad subscript
- 19 redim'd array
- 20 division by zero
- 21 illegal direct
- 22 type mismatch
- 23 string too long
- 24 file data
- 25 formula too complex
- 26 can't continue
- 27 undef'd function
- 28 verify
- 29 load

\*\*\*\*\*

## Adressen der Fehlermeldungen

A32B 9E A1 AC A1 B5 A1 C2 A1  
A330 D0 A1 E2 A1 F0 A1 FF A1  
A33B 10 A2 25 A2 35 A2 3B A2  
A340 4F A2 5A A2 6A A2 72 A2  
A34B 7F A2 90 A2 9D A2 AA A2  
A350 BA A2 CB A2 D5 A2 E4 A2  
A35B ED A2 00 A3 0E A3 1E A3

A360 24 A3 B3 A3

```
*****
A364 0D 4F 4B 0D
A368 00 20 20 45 52 52 4F 52
A370 00 20 49 4E 20 00 0D 0A
A378 52 45 41 44 59 2E 0D 0A
A380 00 0D 0A 42 52 45 41 4B
A388 00 A0
```

Meldungen des Interpreters  
ok  
error  
in  
ready.  
break

```
*****
A38A BA TSX
A38B EB INX
A38C EB INX
A38D EB INX
A38E EB INX
A38F BD 01 01 LDA #0101,X
A392 C9 B1 CMP #B1
A394 D0 21 BNE #A3B7
A396 A5 4A LDA #4A
A398 D0 0A BNE #A3A4
A39A BD 02 01 LDA #0102,X
A39D B5 49 STA #49
A39F BD 03 01 LDA #0103,X
A3A2 B5 4A STA #4A
A3A4 DD 03 01 CMP #0103,X
A3A7 D0 07 BNE #A3B0
A3A9 A5 49 LDA #49
A3AB DD 02 01 CMP #0102,X
A3AE F0 07 BEQ #A3B7
A3B0 BA TIA
A3B1 1B CLC
A3B2 69 12 ADC #12
A3B4 AA TAX
A3B5 D0 D0 BNE #A3BF
A3B7 60 RTS
```

Stapelsuch-Routine für FOR-NEXT  
und 60SUB

FOR-Code

Variablenname vergleichen

Stapelzeiger um 18 erhöhen

nächste Schleife prüfen

```
*****
A3BB 20 0B A4 JSR #A40B
A3BB B5 31 STA #31
A3BD B4 32 STY #32
A3BF 38 SEC
A3C0 A5 5A LDA #5A
A3C2 E5 5F SBC #5F
A3C4 B5 22 STA #22
A3C6 AB TAY
A3C7 A5 5B LDA #5B
A3C9 E5 60 SBC #60
A3CB AA TAX
A3CC EB INX
A3CD 9B TYA
A3CE F0 23 BEQ #A3F3
A3D0 A5 5A LDA #5A
A3D2 38 SEC
A3D3 E5 22 SBC #22
A3D5 B5 5A STA #5A
A3D7 B0 03 BCS #A3DC
A3D9 C6 5B DEC #5B
A3DB 38 SEC
```

Block-Verschiebe-Routine  
prüft auf Platz im Speicher

Einsprungpunkt

\$5F/\$60 Alter Blockanfang  
\$5A/\$5B Altes Blockende + 1  
\$5B/\$59 Neues Blockende + 1

```

A30C A5 5B LDA #5B
A30E E5 22 SBC #22
A3E0 B5 5B STA #5B
A3E2 B0 0B BCS #A3EC
A3E4 C6 59 DEC #59
A3E6 90 04 BCC #A3EC
A3E8 B1 5A LDA (#5A),Y
A3EA 91 5B STA (#5B),Y
A3EC BB DEY
A3ED D0 F9 BNE #A3E8
A3EF B1 5A LDA (#5A),Y
A3F1 91 5B STA (#5B),Y
A3F3 C6 5B DEC #5B
A3F5 C6 59 DEC #59
A3F7 CA DEI
A3FB D0 F2 BNE #A3EC
A3FA 60 RTS

```

```

***** Prufung auf Platz im Stapel
A3FB 0A ASL A      Akku muß dazu die halbe Zahl an
A3FC 69 3E ADC #3E erforderlichen Platz enthalten
A3FE B0 35 BCS #A435 gibt 'out of memory'
A400 B5 22 STA #22
A402 BA TSX
A403 E4 22 CPX #22
A405 90 2E BCC #A435 gibt 'out of memory'
A407 60 RTS

```

```

***** Schafft Platz im Speicher
A40B C4 34 CPY #34      für Zeileneinfügung und Variablenanlage
A40A 90 2B BCC #A434
A40C D0 04 BNE #A412
A40E C5 33 CMP #33
A410 90 22 BCC #A434
A412 4B PHA
A413 A2 09 LDX #09
A415 9B TYA
A416 4B PHA
A417 B5 57 LDA #57,X    Register für Arithmetik retten
A419 CA DEI
A41A 10 FA BPL #A416
A41C 20 26 B5 JSR #B526 Garbage Collection
A41F A2 F7 LDX #F7
A421 6B PLA
A422 95 61 STA #61,X    Register zurückholen
A424 EB INX
A425 30 FA BMI #A421
A427 6B PLA
A428 AB TAY
A429 6B PLA
A42A C4 34 CPY #34
A42C 90 06 BCC #A434    Ok, fertig
A42E D0 05 BNE #A435    kein Platz, 'out of memory'
A430 C5 33 CMP #33
A432 B0 01 BCS #A435
A434 60 RTS

```

```

A435 A2 10 LDX #10      Fehlernummer für 'out of memory'

```

```

***** Fehlereinsprung

```



A437	6C 00 03	JMP (\$0300)	JMP \$E38B, zum BASIC-Warnestart
*****			Fehlermeldung ausgeben
A43A	BA	TXA	Fehlernummer in X-Register, 1 bis 30
A43B	0A	ASL A	
A43C	AA	TAX	
A43D	BD 26 A3	LDA \$A326,X	
A440	85 22	STA \$22	
A442	BD 27 A3	LDA \$A327,X	Adresse der Fehlermeldung holen
A445	85 23	STA \$23	
A447	20 CC FF	JSR \$FFCC	CLRCH aktive I/O Kanäle rucksetzen
A44A	A9 00	LDA #\$00	
A44C	85 13	STA \$13	I/O Flag rucksetzen
A44E	20 D7 AA	JSR \$AAD7	(CR) und (LF) ausgeben
A451	20 45 AB	JSR \$AB45	'?' ausgeben
A454	A0 00	LDY #\$00	
A456	B1 22	LDA (\$22),Y	Text der Fehlermeldung
A458	48	PHA	
A459	29 7F	AND #\$7F	
A45B	20 47 AB	JSR \$AB47	Fehlermeldung ausgeben
A45E	C8	INY	
A45F	68	PLA	
A460	10 F4	BPL \$A456	
A462	20 7A A6	JSR \$A67A	BASIC-Zeiger initialisieren, CONT sperren
A465	A9 69	LDA #\$69	
A467	A0 A3	LDY #\$A3	Zeiger A/Y auf 'error'
A469	20 1E AB	JSR \$AB1E	String ausgeben
A46C	A4 3A	LDY \$3A	Programmmodus ?
A46E	C8	INY	
A46F	F0 03	BEQ \$A474	nein
A471	20 C2 BD	JSR \$BDC2	'in Zeilennummer' ausgeben
A474	A9 76	LDA #\$76	
A476	A0 A3	LDY #\$A3	Zeiger auf 'ready'
A478	20 1E AB	JSR \$AB1E	String ausgeben
A47B	A9 80	LDA #\$80	
A47D	20 90 FF	JSR \$FF90	Direkt-Modus Flag setzen
*****			Eingabe-Warteschleife
A480	6C 02 03	JMP (\$0302)	JMP \$A483
A483	20 60 A5	JSR \$A560	BASIC-Zeile in Eingabepuffer holen
A486	B6 7A	STX \$7A	
A488	84 7B	STY \$7B	CHRGET Zeiger auf Eingabepuffer
A48A	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
A48D	AA	TAX	
A48E	F0 F0	BEQ \$A480	Puffer leer, dann weiter warten
A490	A2 FF	LDX \$FF	
A492	B6 3A	STX \$3A	Kennzeichen für Direkt-Modus
A494	90 06	BCC \$A49C	Ziffer, dann als BASIC-Zeile einfügen
A496	20 79 A5	JSR \$A579	BASIC-Zeile in Interpreter-Kode wandeln
A499	4C E1 A7	JMP \$A7E1	Befehl ausführen
*****			Loschen und Einfügen von Programmzeilen
A49C	20 6B A9	JSR \$A96B	Zeilennummer in Adressformat umwandeln \$14/\$15
A49F	20 79 A5	JSR \$A579	BASIC-Zeile in Interpreter-Kode wandeln
A4A2	84 0B	STY \$0B	Zeiger in Eingabepuffer
A4A4	20 13 A6	JSR \$A613	Adresse der BASIC-Zeile berechnen
A4A7	90 44	BCC \$A4ED	nicht vorhanden, dann löschen übergehen
*****			Programmzeile löschen
A4A9	A0 01	LDY #\$01	

A4AB	B1 5F	LDA (\$5F),Y	
A4AD	B5 23	STA \$23	
A4AF	A5 2D	LDA \$2D	
A4B1	B5 22	STA \$22	
A4B3	A5 60	LDA \$60	
A4B5	B5 25	STA \$25	
A4B7	A5 5F	LDA \$5F	
A4B9	BB	DEY	
A4BA	F1 5F	SBC (\$5F),Y	
A4BC	1B	CLC	
A4BD	65 2D	ADC \$2D	
A4BF	B5 2D	STA \$2D	
A4C1	B5 24	STA \$24	
A4C3	A5 2E	LDA \$2E	
A4C5	69 FF	ADC \$FF	
A4C7	B5 2E	STA \$2E	
A4C9	E5 60	SBC \$60	
A4CB	AA	TAX	
A4CC	3B	SEC	
A4CD	A5 5F	LDA \$5F	
A4CF	E5 2D	SBC \$2D	
A4D1	AB	TAY	
A4D2	B0 03	BCS \$A4D7	
A4D4	EB	INX	
A4D5	C6 25	DEC \$25	
A4D7	1B	CLC	
A4D8	65 22	ADC \$22	
A4DA	90 03	BCC \$A4DF	
A4DC	C6 23	DEC \$23	
A4DE	1B	CLC	
A4DF	B1 22	LDA (\$22),Y	Verschiebeschleife
A4E1	91 24	STA (\$24),Y	
A4E3	CB	INY	
A4E4	D0 F9	BNE \$A4DF	
A4E6	E6 23	INC \$23	
A4E8	E6 25	INC \$25	
A4EA	CA	DEY	
A4EB	D0 F2	BNE \$A4DF	
*****			
A4ED	20 59 A6	JSR \$A659	Programmzeile einfügen
A4F0	20 33 A5	JSR \$A533	CLR-Befehl
A4F3	AD 00 02	LDA \$0200	Programmzeilen neu binden
A4F6	F0 BB	BEQ \$A4B0	Zeichen im Puffer ?
A4F8	1B	CLC	nein, dann zur Eingabe-Warteschleife
A4F9	A5 2D	LDA \$2D	
A4FB	B5 5A	STA \$5A	
A4FD	65 0B	ADC \$0B	
A4FF	B5 5B	STA \$5B	
A501	A4 2E	LDY \$2E	
A503	B4 5B	STY \$5B	
A505	90 01	BCC \$A50B	
A507	CB	INY	
A508	B4 59	STY \$59	
A50A	20 BB A3	JSR \$A3B8	BASIC-Zeilen verschieben
A50D	A5 14	LDA \$14	
A50F	A4 15	LDY \$15	
A511	BD FE 01	STA \$01FE	
A514	BC FF 01	STY \$01FF	
A517	A5 31	LDA \$31	

A519	A4 32	LDY #32	
A51B	85 2D	STA #2D	
A51D	84 2E	STY #2E	
A51F	A4 0B	LDY #0B	
A521	8B	DEY	
A522	B9 FC 01	LDA #01FC,Y	Zeichen aus Eingabepuffer
A525	91 5F	STA (\$5F),Y	ins Programm kopieren
A527	8B	DEY	
A528	10 FB	BPL #A522	
A52A	20 59 A6	JSR #A659	CLR-Befehl
A52D	20 33 A5	JSR #A533	Programmzeilen neu binden
A530	4C 80 A4	JMP #A480	zur Eingabe-Warteschleife

\*\*\*\*\* BASIC-Programmzeilen neu binden

A533	A5 2B	LDA #2B
A535	A4 2C	LDY #2C
A537	85 22	STA #22
A539	84 23	STY #23
A53B	1B	CLC
A53C	A0 01	LDY #01
A53E	B1 22	LDA (\$22),Y
A540	F0 1D	BEQ #A55F
A542	A0 04	LDY #04
A544	CB	INY
A545	B1 22	LDA (\$22),Y
A547	D0 FB	BNE #A544
A549	CB	INY
A54A	9B	TYA
A54B	65 22	ADC #22
A54D	AA	TAX
A54E	A0 00	LDY #00
A550	91 22	STA (\$22),Y
A552	A5 23	LDA #23
A554	69 00	ADC #00
A556	CB	INY
A557	91 22	STA (\$22),Y
A559	86 22	STX #22
A55B	85 23	STA #23
A55D	90 DD	BCC #A53C
A55F	60	RTS

\*\*\*\*\* Eingabe einer Zeile

A560	A2 00	LDX #00	
A562	20 12 E1	JSR #E112	ein Zeichen holen
A565	C9 0D	CMP #0D	RETURN-Taste ?
A567	F0 0D	BEQ #A576	ja, dann Eingabe beenden
A569	9D 00 02	STA #0200,X	Zeichen in Eingabepuffer speichern
A56C	EB	INX	
A56D	E0 59	CPI #59	89. Zeichen ?
A56F	90 F1	BCC #A562	nein, weitere Zeichen holen
A571	A2 17	LDX #17	Nummer für 'string too long'
A573	4C 37 A4	JMP #A437	Fehlermeldung ausgeben
A576	4C CA AA	JMP #AACA	Puffer mit #0 abschließen, CR ausgeben

\*\*\*\*\* Umwandlung einer Zeile in Interpreter-Kode

A579	6C 04 03	JMP (#0304)	JMP #A57C
A57C	A6 7A	LDX #7A	
A57E	A0 04	LDY #04	
A580	84 0F	STY #0F	Flag für Hochkomma
A582	BD 00 02	LDA #0200,X	Zeichen aus Puffer holen

ASB5	10 07	BPL \$A5B8	kein BASIC-Kode ?
ASB7	C9 FF	CMP #\$FF	Kode für P1 ?
ASB9	F0 3E	BEQ \$ASC9	
ASBB	E8	INX	
ASBC	D0 F4	BNE \$A5B2	
ASBE	C9 20	CMP #\$20	' ' Leerzeichen
ASB0	F0 37	BEQ \$ASC9	
ASB2	B5 08	STA #08	
ASB4	C9 22	CMP #\$22	"" Hochkomma
ASB6	F0 56	BEQ \$A5EE	
ASB8	24 0F	BIT #0F	
ASBA	70 2D	BVS \$ASC9	
ASBC	C9 3F	CMP #\$3F	'?' Fragezeichen
ASBE	D0 04	BNE \$A5A4	
ASB0	A9 99	LDA #\$99	durch Kode für PRINT ersetzen
ASB2	D0 25	BNE \$ASC9	
ASB4	C9 30	CMP #\$30	'0'
ASB6	90 04	BCC \$A5AC	kleiner ?
ASB8	C9 3C	CMP #\$3C	
ASBA	90 1D	BCC \$ASC9	
ASBC	B4 71	STY \$71	
ASBE	A0 00	LDY #00	
ASB0	B4 08	STY #08	
ASB2	B8	DEY	
ASB3	B6 7A	STX \$7A	
ASB5	CA	DEX	
ASB6	CB	INX	
ASB7	E8	INX	
ASBB	BD 00 02	LDA \$0200,X	Zeichen im Puffer
ASBB	38	SEC	
ASBC	F9 9E A0	SBC \$A09E,Y	mit Befehlsworten in Tabelle vergleichen
ASBF	F0 F5	BEQ \$A5B6	
ASC1	C9 B0	CMP #\$B0	
ASC3	D0 30	BNE \$A5F5	
ASC5	05 08	ORA #08	gefunden, BASIC-Kode gleich Zähler +\$B0
ASC7	A4 71	LDY \$71	
ASC9	E8	INX	
ASCA	CB	INX	
ASCB	99 FB 01	STA \$01FB,Y	BASIC-Kode speichern
ASCE	B9 FB 01	LDA \$01FB,Y	und Statusregister setzen
ASD1	F0 36	BEQ \$A609	Ende, dann fertig
ASD3	38	SEC	
ASD4	E9 3A	SBC #\$3A	':'
ASD6	F0 04	BEQ \$A5DC	Trennzeichen ?
ASD8	C9 49	CMP #\$49	DATA-Kode ?
ASDA	D0 02	BNE \$A5DE	
ASDC	B5 0F	STA #0F	
ASDE	38	SEC	
ASDF	E9 55	SBC #\$55	REM-Kode ?
ASE1	D0 9F	BNE \$A5B2	
ASE3	B5 08	STA #08	
ASE5	BD 00 02	LDA \$0200,X	
ASE8	F0 DF	BEQ \$ASC9	
ASEA	C5 08	CMP #08	
ASEC	F0 DB	BEQ \$ASC9	
ASEE	CB	INX	
ASEF	99 FB 01	STA \$01FB,Y	
ASF2	E8	INX	
ASF3	D0 F0	BNE \$A5E5	
ASF5	A6 7A	LDX #7A	

A5F7	E6 0B	INC \$0B	
A5F9	C8	INY	
A5FA	B9 9D A0	LDA \$A09D,Y	
A5FD	10 FA	BPL \$A5F9	mit Tabelle vergleichen
A5FF	B9 9E A0	LDA \$A09E,Y	
A602	D0 B4	BNE \$A3B8	
A604	BD 00 02	LDA \$0200,X	
A607	10 BE	BPL \$A5C7	
A609	99 FD 01	STA \$01FD,Y	
A60C	C6 7B	DEC \$7B	
A60E	A9 FF	LDA \$FF	Zeiger auf Eingabepuffer - 1
A610	B5 7A	STA \$7A	
A612	60	RTS	
***** Startadresse einer Programmzeile berechnen			
A613	A5 2B	LDA \$2B	
A615	A6 2C	LDX \$2C	Zeiger auf Programmstart
A617	A0 01	LDY \$01	
A619	85 5F	STA \$5F	
A61B	B6 60	STX \$60	
A61D	B1 5F	LDA (\$5F),Y	Link-Adresse high
A61F	F0 1F	BEQ \$A640	gleich null, dann Ende, nicht gefunden
A621	C8	INY	
A622	C8	INY	
A623	A5 15	LDA \$15	gesuchte Zeilennummer high
A625	D1 5F	CMR (\$5F),Y	mit augenblicklicher vergleichen
A627	90 1B	BCC \$A641	kleiner, dann nicht gefunden
A629	F0 03	BEQ \$A62E	gleich, dann Nummer low prüfen
A62B	8B	DEY	
A62C	D0 09	BNE \$A637	unbedingter Sprung
A62E	A5 14	LDA \$14	
A630	8B	DEY	
A631	D1 5F	CMR (\$5F),Y	Zeilennummer low vergleichen
A633	90 0C	BCC \$A641	kleiner
A635	F0 0A	BEQ \$A641	oder gleich C=1
A637	8B	DEY	
A63B	B1 5F	LDA (\$5F),Y	
A63A	AA	TAX	
A63B	8B	DEY	Adresse der nächsten Programmzeile
A63C	B1 5F	LDA (\$5F),Y	
A63E	B0 D7	BCS \$A617	weiter suchen
A640	1B	CLC	
A641	60	RTS	
***** BASIC-Befehl NEW			
A642	D0 FD	BNE \$A641	
A644	A9 00	LDA \$00	
A646	AB	TAY	
A647	91 2B	STA (\$2B),Y	
A649	C8	INY	Zweimal \$00 an Programmstart
A64A	91 2B	STA (\$2B),Y	
A64C	A5 2B	LDA \$2B	
A64E	1B	CLC	
A64F	69 02	ADC \$02	
A651	85 2D	STA \$2D	Variablenstart = Programmstart + 2
A653	A5 2C	LDA \$2C	
A655	69 00	ADC \$00	
A657	B5 2E	STA \$2E	
A659	20 BE A6	JSR \$A6BE	CHRGET-Zeiger auf Programmstart
A65C	A9 00	LDA \$00	

```

***** BASIC-Befehl CLR
A65E D0 2D BNE #A68D
A660 20 E7 FF JSR $FFE7 CLALL alle I/O Kanäle rucksetzen
A663 A5 37 LDA $37
A665 A4 38 LDY #38
A667 85 33 STA $33 String-Start auf BASIC-RAM Ende
A669 84 34 STY $34
A66B A5 2D LDA $2D
A66D A4 2E LDY $2E
A66F 85 2F STA $2F Variablen-Ende = Variablenanfang
A671 84 30 STY $30
A673 85 31 STA $31
A675 84 32 STY $32
A677 20 1D AB JSR $AB1D RESTORE Befehl
A67A A2 19 LDI #19
A67C 86 16 STX $16 String-Descriptor Index rucksetzen
A67E 68 PLA
A67F AB TAY
A680 68 PLA
A681 A2 FA LDI #FA
A683 9A TIX Stackpointer initialisieren
A684 48 PHA
A685 98 TYA
A686 48 PHA
A687 A9 00 LDA #00
A689 85 3E STA $3E CONY sperren
A68B 85 10 STA $10
A68D 60 RTS

***** Programmzeiger auf BASIC-Start
A68E 18 CLC
A68F A5 2B LDA $2B
A691 69 FF ADC #FF BASIC-Start
A693 85 7A STA $7A
A695 A5 2C LDA $2C
A697 69 FF ADC #FF minus 1
A699 85 7B STA $7B
A69B 60 RTS

***** BASIC Befehl LIST
A69C 90 06 BCC $A6A4 Ziffer ? (Zeilennummer)
A69E F0 04 BEQ $A6A4 nur LIST ?
A6A0 C9 AB CMP #AB Kode für '-'
A6A2 D0 E9 BNE $A6BD anderer Kode, dann SYNTAX ERROR
A6A4 20 6B A9 JSR $A96B Zeilennummer holen
A6A7 20 13 A6 JSR $A613 Startadresse der Programmzeile berechnen
A6AA 20 79 00 JSR $0079 CHRGET letztes Zeichen holen
A6AD F0 0C BEQ $A6BB keine Zeilennummer
A6AF C9 AB CMP #AB Kode für '-'
A6B1 D0 BE BNE $A641 nein, SYNTAX ERROR
A6B3 20 73 00 JSR $0073 CHRGET nächstes Zeichen holen
A6B6 20 6B A9 JSR $A96B Zeilennummer holen
A6B9 D0 86 BNE $A641
A6BB 68 PLA
A6BC 68 PLA
A6BD A5 14 LDA $14
A6BF 05 15 ORA $15 zweite Zeilennummer gleich null ?
A6C1 D0 06 BNE $A6C9 nein
A6C3 A9 FF LDA #FF
A6C5 85 14 STA $14

```

A6C7	85 15	STA \$15	bis zum Ende listen
A6C9	A0 01	LDY #01	
A6CB	B4 0F	STY \$0F	
A6CD	B1 5F	LDA (\$5F),Y	Linkadresse high
A6CF	F0 43	BEQ \$A714	ja, fertig
A6D1	20 2C AB	JSR \$AB2C	prüft auf Stop-Taste
A6D4	20 D7 AA	JSR \$AAD7	'CR' ausgeben, neue Zeile
A6D7	CB	INY	
A6DB	B1 5F	LDA (\$5F),Y	
A6DA	AA	TAX	
A6DB	CB	INY	Zeilennummer holen
A6DC	B1 5F	LDA (\$5F),Y	
A6DE	C5 15	CMP \$15	
A6E0	D0 04	BNE \$A6E6	mit Endnummer vergleichen
A6E2	E4 14	CPX \$14	
A6E4	F0 02	BEQ \$A6E8	
A6E6	80 2C	BCS \$A714	größer, dann fertig
A6E8	B4 49	STY \$49	
A6EA	20 CD BD	JSR \$BDCD	Zeilennummer ausgeben
A6ED	A9 20	LDA #20	' ' Leerzeichen
A6EF	A4 49	LDY \$49	
A6F1	29 7F	AND #\$7F	
A6F3	20 47 AB	JSR \$AB47	Zeichen ausgeben
A6F6	C9 22	CMP #22	'.' Hochkomma ?
A6F8	D0 06	BNE \$A700	
A6FA	A5 0F	LDA \$0F	
A6FC	49 FF	EDR #\$FF	Hochkommaflag umdrehen
A6FE	85 0F	STA \$0F	
A700	CB	INY	kein Zeilenende nach 255 Zeichen ?
A701	F0 11	BEQ \$A714	dann aufhören
A703	B1 5F	LDA (\$5F),Y	Zeichen holen
A705	D0 10	BNE \$A717	kein Zeilenende, dann listen
A707	AB	TAY	
A708	B1 5F	LDA (\$5F),Y	
A70A	AA	TAX	Startadresse der nächsten Zeile
A70B	CB	INY	
A70C	B1 5F	LDA (\$5F),Y	
A70E	B6 5F	STX \$5F	
A710	85 60	STA \$60	als Zeiger merken
A712	D0 B5	BNE \$A6C9	weiter machen
A714	4C B6 E3	JMP \$E386	zum BASIC-Warrestart
*****			BASIC-Kode in Klartext umwandlen
A717	6C 06 03	JMP (\$0306)	JMP \$A71A
A71A	10 D7	BPL \$A6F3	kein Interpreter-Kode, so ausgeben
A71C	C9 FF	CMP #\$FF	Kode für P1
A71E	F0 D3	BEQ \$A6F3	so ausgeben
A720	24 0F	BIT 0F	Hochkommaodus ?
A722	30 CF	BMI \$A6F3	dann Zeichen so ausgeben
A724	38	SEC	
A725	E9 7F	SBC #\$7F	Offset abziehen
A727	AA	TAX	Kode nach X
A728	B4 49	STY \$49	Zeichenzeiger merken
A72A	A0 FF	LDY #\$FF	
A72C	CA	DEX	erstes Befehlswort ?
A72D	F0 08	BEQ \$A737	
A72F	CB	INY	
A730	B9 9E A0	LDA \$A09E,Y	Offset für Xtes Befehlswort finden
A733	10 FA	BPL \$A72F	
A735	30 F3	BMI \$A72C	Bit 7 gesetzt, nächstes Wort

A737	C8	INY	
A738	B9 9E A0	LDA #A09E,Y	Befehlswort aus Tabelle holen
A73B	30 B2	BMI #A6EF	letzter Buchstabe, dann fertig
A73D	20 47 AB	JSR #AB47	Zeichen ausgeben
A740	D0 F5	BNE #A737	nächsten Buchstaben ausgeben
***** BASIC-Befehl FOR *****			
A742	A9 80	LDA #80	Integer sperren
A744	B5 10	STA #10	
A746	20 A5 A9	JSR #A9A5	LET-Befehl, setzt FOR-Variable
A749	20 8A A3	JSR #A38A	sucht offene FOR-NEXT-Schleife mit gleicher
A74C	D0 05	BNE #A753	nicht gefunden Variabler
A74E	8A	TIA	
A74F	69 0F	ADC #0F	Stapelzeiger erhöhen
A751	AA	TAX	
A752	9A	TIS	
A753	68	PLA	Rücksprungadresse vom Stack holen
A754	68	PLA	
A755	A9 09	LDA #09	
A757	20 FB A3	JSR #A3FB	prüft auf genügend Platz in Stack
A75A	20 06 A9	JSR #A906	sucht nächstes BASIC-Statement
A75D	18	CLC	
A75E	98	TYA	Programmzeiger auf nächsten Befehl
A75F	65 7A	ADC #7A	
A761	48	PHA	und auf Stack speichern
A762	A5 7B	LDA #7B	
A764	69 00	ADC #00	
A766	48	PHA	
A767	A5 3A	LDA #3A	
A769	48	PHA	laufende Zeilennummer auf Stack
A76A	A5 39	LDA #39	
A76C	48	PHA	
A76D	A9 A4	LDA #A4	'TQ' - Kode
A76F	20 FF AE	JSR #AEFF	prüft auf Kode
A772	20 BD AD	JSR #ADBD	prüft ob numerische Variable
A775	20 BA AD	JSR #ADBA	holt numerischen Ausdruck nach FAC
A778	A5 66	LDA #66	
A77A	09 7F	DRA #7F	
A77C	25 62	AND #62	
A77E	B5 62	STA #62	
A780	A9 8B	LDA #8B	
A782	A0 A7	LDY #A7	Rücksprungadresse merken
A784	B5 22	STA #22	
A786	B4 23	STY #23	
A788	4C 43 AE	JMP #AE43	legt Schleifenendwert auf Stack
A78B	A9 BC	LDA #BC	
A78D	A0 B9	LDY #B9	Zeiger auf Konstante i
A78F	20 A2 BB	JSR #BBA2	als Default-STEP-Wert in FAC
A792	20 79 00	JSR #0079	CHRGET letztes Zeichen holen
A795	C9 A9	CMR #A9	'STEP' - Kode
A797	D0 06	BNE #A79F	kein STEP-Wert
A799	20 73 00	JSR #0073	CHRGET nächstes Zeichen holen
A79C	20 8A AD	JSR #ADBA	holt numerischen Ausdruck nach FAC
A79F	20 2B BC	JSR #BC2B	holt Vorzeichen
A7A2	20 38 AE	JSR #AE38	legt Vorzeichen und STEP-Wert auf Stack
A7A5	A5 4A	LDA #4A	
A7A7	48	PHA	Variablenname
A7A8	A5 49	LDA #49	
A7AA	48	PHA	
A7AB	A9 81	LDA #81	und FOR-Kode auf Stack



A7AD	4B	PHA	
*****			Interpreterschleife
A7AE	20 2C AB	JSR \$AB2C	prüft auf Stop-Taste
A7B1	A5 7A	LDA \$7A	
A7B3	A4 7B	LDY \$7B	Programmezeiger
A7B5	C0 02	CPY #\$02	Direkt-Modus ?
A7B7	EA	NDP	
A7B8	F0 04	BEQ \$A7BE	ja
A7BA	B5 3D	STA \$3D	als Zeiger für CONT merken
A7BC	84 3E	STY \$3E	
A7BE	A0 00	LDY #\$00	
A7C0	B1 7A	LDA (\$7A),Y	laufendes Zeichen
A7C2	D0 43	BNE \$AB07	nicht Zeilenende ?
A7C4	A0 02	LDY #\$02	
A7C6	B1 7A	LDA (\$7A),Y	Programmende
A7C8	18	CLC	Flag für END setzen
A7C9	D0 03	BNE \$A7CE	
A7CB	4C 4B AB	JMP \$AB4B	ja dann END ausführen
A7CE	CB	INY	
A7CF	B1 7A	LDA (\$7A),Y	
A7D1	B5 39	STA \$39	
A7D3	CB	INY	laufende Zeilennummer merken
A7D4	B1 7A	LDA (\$7A),Y	
A7D6	B5 3A	STA \$3A	
A7D8	9B	TYA	
A7D9	65 7A	ADC \$7A	Programmezeiger auf Programmzeile setzen
A7DB	85 7A	STA \$7A	
A7DD	90 02	BCC \$A7E1	
A7DF	E6 7B	INC \$7B	
A7E1	6C 0B 03	JMP (\$030B)	JMP \$A7E4 BASIC-Statement ausführen
A7E4	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
A7E7	20 ED A7	JSR \$A7ED	Statement ausführen
A7EA	4C AE A7	JMP \$A7AE	zurück zur Interpreterschleife
*****			BASIC-Statement ausführen
A7ED	F0 3C	BEQ \$AB2B	Zeilenende, dann fertig
A7EF	E9 80	SBC #\$80	Token ?
A7F1	90 11	BCC \$AB04	nein, dann zum LET-Befehl
A7F3	C9 23	CMP #\$23	
A7F5	B0 17	BCS \$AB0E	Funktions-Token oder GO TO
A7F7	0A	ASL A	BASIC-Befehl, Kode mal 2
A7F8	AB	TAY	
A7F9	B9 0D A0	LDA \$A00D,Y	
A7FC	4B	PHA	Befehlsadresse aus Tabelle holen
A7FD	B9 0C A0	LDA \$A00C,Y	
AB00	4B	PHA	als Rücksprungadresse auf Stack
AB01	4C 73 00	JMP \$0073	nächstes Zeichen und Befehl ausführen
AB04	4C A5 A9	JMP \$A9A5	zum LET-Befehl
*****			
AB07	C9 3A	CMP #\$3A	':'
AB09	F0 D6	BEQ \$A7E1	
AB0B	4C 0B AF	JMP \$AF0B	'syntax error'
*****			prüft auf 'GO' 'TO' Kode
AB0E	C9 4B	CMP #\$4B	'GO' (minus \$80)
AB10	D0 F9	BNE \$AB0B	
AB12	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
AB15	A9 A4	LDA \$A4	'TO'

AB17	20 FF AE	JSR \$AEFF	prüft auf Kode
AB1A	4C A0 AB	JMP \$ABA0	zum GOTO-Befehl
*****			BASIC-Befehl RESTORE
AB1D	38	SEC	
AB1E	A5 2B	LDA \$2B	
AB20	E9 01	SBC #01	Programmstart - 1
AB22	A4 2C	LDY \$2C	
AB24	80 01	BCS \$AB27	
AB26	88	DEY	
AB27	85 41	STA \$41	
AB29	84 42	STY \$42	gleich DATA-Zeiger
AB2B	60	RTS	
*****			prüft auf Stop-Taste
AB2C	20 E1 FF	JSR \$FFE1	Stop-Taste abfragen
*****			BASIC-Befehl STOP
AB2F	B0 01	BCS \$AB32	C=1 Flag für STOP
*****			BASIC-Befehl END
AB31	18	CLC	C=0 Flag für END
AB32	D0 3C	BNE \$AB70	
AB34	A5 7A	LDA \$7A	
AB36	A4 7B	LDY \$7B	Programmzeiger
AB38	A6 3A	LDX \$3A	Direkt-Modus ?
AB3A	EB	INX	
AB3B	F0 0C	BEQ \$AB49	ja
AB3D	85 3D	STA \$3D	
AB3F	84 3E	STY \$3E	Zeiger für CONT setzen
AB41	A5 39	LDA \$39	
AB43	A4 3A	LDY \$3A	Nummer der laufenden Zeile
AB45	85 3B	STA \$3B	
AB47	84 3C	STY \$3C	als Zeilennummer für CONT merken
AB49	68	PLA	
AB4A	68	PLA	Rücksprungadresse vom Stack holen
AB4B	A9 B1	LDA #B1	
AB4D	A0 A3	LDY #A3	Zeiger auf 'break'
AB4F	90 03	BCC \$AB54	END Flag ?
AB51	4C 69 A4	JMP \$A469	nein, 'break in xxx' ausgeben
AB54	4C 86 E3	JMP \$E386	zum BASIC-Warstart
*****			BASIC-Befehl CONT
AB57	D0 17	BNE \$AB70	
AB59	A2 1A	LDX #1A	Fehlernummer für 'CAN'T CONTINUE
AB5B	A4 3E	LDY \$3E	CONT gesperrt ?
AB5D	D0 03	BNE \$AB62	nein
AB5F	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
AB62	A5 3D	LDA \$3D	
AB64	85 7A	STA \$7A	Programmzeiger
AB66	84 7B	STY \$7B	
AB68	A5 3B	LDA \$3B	und
AB6A	A4 3C	LDY \$3C	
AB6C	85 39	STA \$39	Zeilennummer setzen
AB6E	84 3A	STY \$3A	
AB70	60	RTS	
*****			BASIC-Befehl RUN
AB71	08	PHP	
AB72	A9 00	LDA #00	

AB74	20 90 FF	JSR \$FF90	Flag für Programm-Modus setzen
AB77	28	PLP	
AB78	D0 03	BNE \$AB7D	weitere Zeichen (Zeilennummer) ?
AB7A	4C 59 A6	JMP \$A659	Programmzeiger auf Programmstart, CLR

AB7D	20 60 A6	JSR \$A660	CLR-Befehl
AB80	4C 97 AB	JMP \$AB97	GOTO-Befehl

\*\*\*\*\* BASIC-Befehl GOSUB

AB83	A9 03	LDA #\$03	
AB85	20 FB A3	JSR \$A3FB	prüft auf genügend Platz im Stack
AB88	A5 7B	LDA \$7B	
AB8A	48	PHA	Programmzeiger
AB8B	A5 7A	LDA \$7A	
AB8D	48	PHA	
AB8E	A5 3A	LDA \$3A	
AB90	48	PHA	und Zeilennummer senken
AB91	A5 39	LDA \$39	
AB93	48	PHA	
AB94	A9 8D	LDA #\$8D	'GOSUB'-Kode auf Stack
AB96	48	PHA	
AB97	20 79 00	JSR \$0079	CHRGOT letztes Zeichen holen
AB9A	20 A0 AB	JSR \$ABA0	GOTO-Befehl
AB9D	4C AE A7	JMP \$A7AE	zur Interpreterschleife

\*\*\*\*\* BASIC-Befehl GOTO

ABA0	20 6B A9	JSR \$A96B	Zeilennummer holen nach \$14/\$15
ABA3	20 09 A9	JSR \$A909	nächsten Zeilenanfang suchen
ABA6	38	SEC	
ABA7	A5 39	LDA \$39	
ABA9	E5 14	SBC \$14	ist Zeilennummer kleiner als laufende Zeile ?
ABAB	A5 3A	LDA \$3A	
ABAD	E5 15	SBC \$15	
ABAF	B0 0B	BCS \$ABBC	nein
ABB1	98	TYA	
ABB2	38	SEC	
ABB3	65 7A	ADC \$7A	
ABB5	A6 7B	LDX \$7B	sucht ab laufender Zeile
ABB7	90 07	BCC \$ABCO	
ABB9	E8	INX	
ABBA	B0 04	BCS \$ABCO	
ABBC	A5 2B	LDA \$2B	sucht ab Programmstart
ABBE	A6 2C	LDX \$2C	
ABCO	20 17 A6	JSR \$A617	sucht Programmzeile
ABC3	90 1E	BCC \$ABE3	nicht gefunden, dann 'undef'd statement'
ABC5	A5 5F	LDA \$5F	
ABC7	E9 01	SBC #\$01	
ABC9	B5 7A	STA \$7A	Programmzeiger auf neue Zeile setzen
ABCB	A5 60	LDA \$60	
ABCD	E9 00	SBC #\$00	
ABCF	B5 7B	STA \$7B	
ABD1	60	RTS	

\*\*\*\*\* BASIC-Befehl RETURN

ABD2	D0 FD	BNE \$ABD1	
ABD4	A9 FF	LDA \$FF	
ABD6	B5 4A	STA \$4A	
ABD8	20 8A A3	JSR \$A38A	nächsten GOSUB-Datensatz im Stack suchen
ABDB	9A	TXS	
ABDC	C9 8D	CMP #\$8D	'GOSUB'-Kode

ABDE	F0 0B	BEQ \$ABEB	gefunden ?
ABE0	A2 0C	LDX #0C	Nummer für 'return without gosub'
AE E2	2C	.BYTE \$2C	
AE E3	A2 11	LDI #02	Nummer für 'undef'd statement'
ABE5	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
ABE8	4C 0B AF	JMP \$AF0B	'syntax error' ausgeben

ABEB	6B	PLA	
ABEC	6B	PLA	
ABED	B5 39	STA \$39	
ABEF	6B	PLA	Zeilennummer von Stack holen
ABF0	B5 3A	STA \$3A	
ABF2	6B	PLA	
ABF3	B5 7A	STA \$7A	
ABF5	6B	PLA	Programmzeiger vom Stack holen
ABF6	B5 7B	STA \$7B	

*****			BASIC-Befehl DATA
ABF8	20 06 A9	JSR \$A906	nächstes Statement suchen
ABF9	9B	TYA	Offset
ABFC	1B	CLC	
ABFD	65 7A	ADC \$7A	
ABFF	B5 7A	STA \$7A	
A901	90 02	BCC \$A905	zu Programmzeiger addieren
A903	E6 7B	INC \$7B	
A905	60	RTS	

*****			Offset des nächsten Trennzeichens finden
A906	A2 3A	LDX #3A	';' Doppelpunkt
A908	2C	.BYTE \$2C	
A909	A2 00	LDX #00	\$0 Zeilenende
A90B	B6 07	STX \$07	
A90D	A0 00	LDY #00	Y enthält Offset
A90F	B4 0B	STY \$0B	
A911	A5 0B	LDA \$0B	
A913	A6 07	LDX \$07	gesuchtes Zeichen
A915	B5 07	STA \$07	
A917	B6 0B	STX \$0B	
A919	B1 7A	LDA (\$7A),Y	Zeichen holen
A91B	F0 EB	BEQ \$A905	Zeilenende, dann fertig
A91D	C5 0B	CMP \$0B	
A91F	F0 E4	BEQ \$A905	
A921	CB	INY	Zeiger erhöhen
A922	C9 22	CMP #22	''' Hochkomma
A924	D0 F3	BNE \$A919	
A926	F0 E9	BEQ \$A911	

*****			BASIC-Befehl IF
A92B	20 9E AD	JSR \$AD9E	FRMEVL Ausdruck berechnen
A92B	20 79 00	JSR \$0079	CHRGOT letztes Zeichen
A92E	C9 89	CMP #89	'GOTO'-Kode
A930	F0 05	BEQ \$A937	ja
A932	A9 A7	LDA #A7	'THEN'-Kode
A934	20 FF AE	JSR \$AEFF	prüft auf Kode
A937	A5 61	LDA \$61	Ergebnis des IF-Ausdrucks
A939	D0 05	BNE \$A940	Ausdruck wahr ?

*****			BASIC-Befehl REM
A93B	20 09 A9	JSR \$A909	nein, nächsten Zeilenanfang suchen
A93E	F0 BB	BEQ \$ABF8	Programmzeiger auf nächste Zeile

A940	20 79 00	JSR \$0079	CHRGET letztes Zeichen holen
A943	B0 03	BCS \$A948	keine Ziffer ?
A945	4C A0 AB	JMP \$ABAO	zum GOTO-Befehl
A948	4C ED A7	JMP \$A7ED	nächsten Befehl dekodieren und ausführen

\*\*\*\*\* BASIC-Befehl ON

A94B	20 9E B7	JSR \$B79E	Byte-Wert (0 bis 255) holen
A94E	48	PHA	Kode merken
A94F	C9 8D	CMP #\$8D	'GOSUB'-Kode
A951	F0 04	BEQ \$A957	ja
A953	C9 89	CMP #\$89	'GOTO'-Kode
A955	D0 91	BNE \$ABE8	nein, dann 'syntax error'
A957	C6 65	DEC \$65	Zähler erniedrigen
A959	D0 04	BNE \$A95F	nach nicht null ?
A95B	68	PLA	ja, Kode zurückholen
A95C	4C EF A7	JMP \$A7EF	und Befehl ausführen

A95F	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
A962	20 6B A9	JSR \$A96B	Zeilennummer holen
A965	C9 2C	CMP #\$2C	',' Komma ?
A967	F0 EE	BEQ \$A957	ja, dann weiter
A969	68	PLA	kein Sprung, Kode zurückholen, fertig
A96A	60	RTS	

\*\*\*\*\* holt Zeilennummer nach \$14/\$15

A96B	A2 00	LDX \$800	
A96D	B6 14	STX \$14	
A96F	B6 15	STX \$15	Vorbesetzung für Zeilennummer gleich 0
A971	B0 F7	BCS \$A96A	keine Ziffer, dann fertig
A973	E9 2F	SBC #\$2F	'0'-1 abziehen, gibt Hexwert
A975	B5 07	STA \$07	merken
A977	A5 15	LDA \$15	
A979	B5 22	STA \$22	
A97B	C9 19	CMP #\$19	Zahl bereits größer oder gleich 6400 ?
A97D	B0 D4	BCS \$A953	dann 'syntax error'
A97F	A5 14	LDA \$14	
A981	0A	ASL A	
A982	26 22	ROL \$22	Zahl *10 (= *2+2+Zahl * 2)
A984	0A	ASL A	
A985	26 22	ROL \$22	
A987	65 14	ADC \$14	
A989	B5 14	STA \$14	
A98B	A5 22	LDA \$22	
A98D	65 15	ADC \$15	
A98F	B5 15	STA \$15	
A991	06 14	ASL \$14	
A993	26 15	ROL \$15	
A995	A5 14	LDA \$14	
A997	65 07	ADC \$07	und Einerziffer addieren
A999	B5 14	STA \$14	
A99B	90 02	BCC \$A99F	
A99D	E6 15	INC \$15	
A99F	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
A9A2	4C 71 A9	JMP \$A971	und weiter machen

\*\*\*\*\* BASIC-Befehl LET

A9A5	20 8B B0	JSR \$B0BB	sucht Variable
A9A8	B5 49	STA \$49	
A9AA	B4 4A	STY \$4A	Variablenadresse merken
A9AC	A9 B2	LDA \$B2	'=' - Kode

A9AE	20 FF AE	JSR \$AEFF	prüft auf Kode
A9B1	A5 0E	LDA \$0E	Integer-Flag
A9B3	48	PHA	
A9B4	A5 0D	LDA \$0D	und Typ-Flag (String/numerisch) merken
A9B5	48	PHA	
A9B7	20 9E AD	JSR \$AD9E	FRMEVL Ausdruck holen
A9BA	68	PLA	
A9BB	2A	ROL A	Typ-Flag zuruckholen
A9BC	20 90 AD	JSR \$AD90	und auf richtigen Typ prüfen
A9BF	D0 18	BNE \$A9D9	
A9C1	68	PLA	
A9C2	10 12	BPL \$A9D5	REAL ?

\*\*\*\*\* Wertzuweisung INTEGER

A9C4	20 18 BC	JSR \$BC18	FAC runden
A9C7	20 BF 91	JSR \$B1BF	und nach INTEGER wandeln
A9CA	A0 00	LDY #00	
A9CC	A5 64	LDA \$64	
A9CE	91 49	STA (\$49),Y	Wert in Variable bringen
A9D0	C8	INY	
A9D1	A5 65	LDA \$65	
A9D3	91 49	STA (\$49),Y	
A9D5	60	RTS	

\*\*\*\*\* Wertzuweisung REAL

A9D6	4C D0 BB	JMP \$BBD0	FAC nach Variable bringen
------	----------	------------	---------------------------

\*\*\*\*\* Wertzuweisung String

A9D9	68	PLA	
A9DA	A4 4A	LDY \$4A	Variablenadresse high
A9DC	C0 BF	CPY #BF	ist Variable Tif ?
A9DE	D0 4C	BNE \$AA2C	nein
A9E0	20 A6 B6	JSR \$B6A6	FRESTR
A9E3	C9 06	CMP #06	Stringlänge gleich 6
A9E5	D0 3D	BNE \$AA24	nein, 'illegal quantity'
A9E7	A0 00	LDY #00	
A9E9	B4 61	STY \$61	
A9EB	B4 66	STY \$66	
A9ED	B4 71	STY \$71	
A9EF	20 1D AA	JSR \$AA1D	prüft nächstes Zeichen auf Ziffer
A9F2	20 E2 BA	JSR \$BAE2	FAC = FAC + 10
A9F5	E6 71	INC \$71	Stellenzähler erhöhen
A9F7	A4 71	LDY \$71	
A9F9	20 1D AA	JSR \$AA1D	prüft nächstes Zeichen auf Ziffer
A9FC	20 0C BC	JSR \$BC0C	FAC nach ARG kopieren
A9FF	AA	TAX	
AA00	F0 05	BEQ \$AA07	FAC gleich null ?
AA02	EB	INX	
AA03	BA	TXA	
AA04	20 ED BA	JSR \$BAED	FAC = FAC + ARG
AA07	A4 71	LDY \$71	Stellenzähler
AA09	C8	INY	erhöhen
AA0A	C0 06	CPY #06	schon 6 Stellen ?
AA0C	D0 DF	BNE \$A9ED	
AA0E	20 E2 BA	JSR \$BAE2	FAC = FAC + 10
AA11	20 9B BC	JSR \$BC9B	FAC rechtsbündig machen
AA14	A6 64	LDX \$64	
AA16	A4 63	LDY \$63	eingeegebene Uhrzeit
AA18	A5 65	LDA \$65	
AA1A	4C DB FF	JMP \$FFDB	Time setzen

*****				Zeichen auf Ziffer prüfen
AA1D	B1 22	LDA (\$22),Y		Zeichen
AA1F	20 B0 00	JSR \$00B0		auf Ziffer prüfen
AA22	90 03	BCC \$AA27		
AA24	4C 4B B2	JMP \$B24B		'illegal quantity'
AA27	E9 2F	SBC \$B2F		von ASCII nach hex umwandeln
AA29	4C 7E BD	JMP \$BD7E		in FAC und ARG übertragen
*****				Wertzuweisung an normalen String
AA2C	A0 02	LDY \$02		
AA2E	B1 64	LDA (\$64),Y		Stringadresse high
AA30	C5 34	CMP \$34		mit Stringanfangsadresse vergleichen
AA32	90 17	BCC \$AA4B		kleiner, String steht innerhalb Programm
AA34	D0 07	BNE \$AA3D		
AA36	8B	DEY		
AA37	B1 64	LDA (\$64),Y		Stringadresse low
AA39	C5 33	CMP \$33		vergleichen
AA3B	90 0E	BCC \$AA4B		String in Programm
AA3D	A4 65	LDY \$65		
AA3F	C4 2E	CPY \$2E		
AA41	90 0B	BCC \$AA4B		
AA43	D0 0D	BNE \$AA52		
AA45	A5 64	LDA \$64		
AA47	C5 2D	CMP \$2D		
AA49	B0 07	BCS \$AA52		
AA4B	A5 64	LDA \$64		
AA4D	A4 65	LDY \$65		
AA4F	4C 6B AA	JMP \$AA6B		
AA52	A0 00	LDY \$00		
AA54	B1 64	LDA (\$64),Y		Länge des Strings
AA56	20 75 B4	JSR \$B475		prüft Speicherplatz, setzt Stringzeiger
AA59	A5 50	LDA \$50		
AA5B	A4 51	LDY \$51		
AA5D	B5 6F	STA \$6F		
AA5F	B4 70	STY \$70		
AA61	20 7A B6	JSR \$B67A		String in Stringbereich übertragen
AA64	A9 61	LDA \$61		
AA66	A0 00	LDY \$00		
AA6B	B5 50	STA \$50		
AA6A	B4 51	STY \$51		
AA6C	20 DB B6	JSR \$B6DB		Descriptor aus Stringstack löschen
AA6F	A0 00	LDY \$00		
AA71	B1 50	LDA (\$50),Y		Länge
AA73	91 49	STA (\$49),Y		
AA75	C8	INY		
AA76	B1 50	LDA (\$50),Y		Adresse low
AA7B	91 49	STA (\$49),Y		
AA7A	C8	INY		
AA7B	B1 50	LDA (\$50),Y		und Adresse high
AA7D	91 49	STA (\$49),Y		in Variable bringen
AA7F	60	RTS		
*****				BASIC-Befehl PRINT#
AA80	20 B6 AA	JSR \$AAB6		CMD-Befehl
AA83	4C B5 AB	JMP \$ABB5		und CLRCH
*****				BASIC-Befehl CMD
AA86	20 9E B7	JSR \$B79E		holt Byte-Ausdruck
AA89	F0 05	BEQ \$AA90		
AA8B	A9 2C	LDA \$B2C		','

AA8D	20 FF AE	JSR \$AEFF	prüft auf Komma
AA90	08	PHP	
AA91	86 13	STX \$13	Nummer des Ausgabegeräts merken
AA93	20 18 E1	JSR \$E118	CKOUT, Ausgabegerät setzen
AA96	28	PLP	
AA97	4C A0 AA	JMP \$AAA0	zum PRINT-Befehl
AA9A	20 21 AB	JSR \$AB21	String drucken
AA9D	20 79 00	JSR \$0079	CHRGET letztes Zeichen

\*\*\*\*\* BASIC-Befehl PRINT

AAA0	F0 35	BEQ \$AAD7	
AAA2	F0 43	BEQ \$AAE7	
AAA4	C9 A3	CMP #\$A3	'TAB'(-Kode
AAA6	F0 50	BEQ \$AAF8	
AAA8	C9 A6	CMP #\$A6	'SPC'(-Kode
AAAA	18	CLC	
AAAB	F0 4B	BEQ \$AAF8	
AAAD	C9 2C	CMP #\$2C	
AAAF	F0 37	BEQ \$AAE9	
AAB1	C9 3B	CMP #\$3B	
AAB3	F0 5E	BEQ \$AB13	
AAB5	20 9E AD	JSR \$AD9E	FRMEVL Term holen
AAB8	24 0D	BIT \$0D	Typflag
AABA	30 DE	BMI \$AA9A	String ?
AABC	20 DD BD	JSR \$BDDD	FAC in ASCII-String umwandeln
AABF	20 B7 B4	JSR \$B4B7	Stringparameter holen
AAC2	20 21 AB	JSR \$AB21	String drucken
AAC5	20 3B AB	JSR \$AB3B	Cursor right bzw. Leerzeichen ausgeben
AACB	D0 D3	BNE \$AA9D	weiter machen
AACA	A9 00	LDA #\$00	Eingabepuffer
AACC	9D 00 02	STA \$0200,X	mit \$0 abschließen
AACF	A2 FF	LDX #\$FF	
AAD1	A0 01	LDY #\$01	Zeiger auf Eingabepuffer setzen
AAD3	A5 13	LDA \$13	Nummer des Ausgabegeräts
AAD5	D0 10	BNE \$AAE7	
AAD7	A9 0D	LDA #\$0D	'CR' carriage return
AAD9	20 47 AB	JSR \$AB47	ausgeben
AADC	24 13	BIT \$13	logische Filenummer
AADE	10 05	BPL \$AAE5	kleiner 128 ?
AAE0	A9 0A	LDA #\$0A	'LF' line feed
AAE2	20 47 AB	JSR \$AB47	ausgeben
AAE5	49 FF	EOR #\$FF	
AAE7	60	RTS	

AAEB	38	SEC	Zehner-Tabulator mit Komma
AAE9	20 F0 FF	JSR \$FFFF0	Cursorposition holen
AAEC	98	TYA	
AAED	38	SEC	
AAEE	E9 0A	SBC #\$0A	10 abziehen
AAF0	B0 FC	BCS \$AAEE	nicht negativ ?
AAF2	49 FF	EOR #\$FF	invertieren
AAF4	69 01	ADC #\$01	eins addieren
AAF6	D0 16	BNE \$AB0E	

\*\*\*\*\* TAB( (C=1) und SPC( (C=0)

AAF8	08	PHP	Flag merken
AAF9	38	SEC	
AAFA	20 F0 FF	JSR \$FFFF0	Cursorposition holen
AAFD	B4 09	STY \$09	und merken
AAFF	20 9B B7	JSR \$B79B	Byte-Wert holen
AB02	C9 29	CMP #\$29	')' Klammer zu ?



AB04	D0 59	BNE \$AB5F	nein, 'syntax error'
AB06	28	PLP	
AB07	90 06	BCC \$AB0F	zu SPC(
AB09	8A	TXA	TAB-Wert in Akku
AB0A	E5 09	SBC \$09	mit Cursorposition vergleichen
AB0C	90 05	BCC \$AB13	Wert kleiner Cursor-Position, dann fertig
AB0E	AA	TAX	
AB0F	EB	INX	
AB10	CA	DEX	
AB11	D0 06	BNE \$AB19	
AB13	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
AB16	4C A2 AA	JMP \$AAA2	und weiter machen
AB19	20 3B AB	JSR \$AB3B	Cursor right bzw. Leerzeichen ausgeben
AB1C	D0 F2	BNE \$AB10	zum Schleifenanfang
*****			String ausgeben
AB1E	20 B7 B4	JSR \$B4B7	Stringparameter holen
AB21	20 A6 B6	JSR \$B6A6	FRESTR
AB24	AA	TAX	Stringlänge
AB25	A0 00	LDY #\$00	
AB27	EB	INX	
AB28	CA	DEX	
AB29	F0 BC	BEQ \$AAE7	String zu Ende ?
AB2B	B1 C2	LDA (\$22),Y	Zeichen des Strings
AB2D	20 47 AB	JSR \$AB47	ausgeben
AB30	CB	INY	
AB31	C9 0D	CMP #\$0D	'CR' carriage return
AB33	D0 F3	BNE \$AB2B	nein, weiter
AB35	20 E5 AA	JSR \$AAE5	Fehler ' Test auf LF-Ausgabe JSR \$AADC
AB3B	4C 2B AB	JMP \$AB2B	und weiter machen
*****			Ausgabe eines Leerzeichens bzw. Cursor right
AB3B	A5 13	LDA \$13	Ausgabe in File ?
AB3D	F0 03	BEQ \$AB42	nein Bildschirm, dann Cursor right
AB3F	A9 20	LDA \$020	' ' Leerzeichen
AB41	2C	.BYTE \$2C	
AB42	A9 1D	LDA \$01D	Cursor right
AB44	2C	.BYTE \$2C	
AB3F	A9 3F	LDA \$03F	'?' Fragezeichen
AB47	20 0C E1	JSR \$E10C	ausgeben
AB4A	29 FF	AND \$FF	Flags setzen
AB4C	60	RTS	
*****			Fehlerbehandlung bei Eingabe
AB4D	A5 11	LDA \$11	Flag for INPUT / GET / READ
AB4F	F0 11	BEQ \$AB62	INPUT
AB51	30 04	BNL \$AB57	READ
AB53	A0 FF	LDY \$FF	
AB55	D0 04	BNE \$AB5B	GET
*****			Fehler bei READ
AB57	A5 3F	LDA \$3F	DATA-Zeilennummer
AB59	A4 40	LDY \$40	
*****			Fehler bei GET
AB5B	B5 39	STA \$39	
AB5D	B4 3A	STY \$3A	gleich Zeilennummer des Fehlers
AB5F	4C 0B AF	JMP \$AF0B	'syntax error'

```

***** Fehler bei INPUT
AB62 A5 13 LDA $13 Nummer des Eingabegeräts
AB64 F0 05 BEQ $AB6B Tastatur ?
AB66 A2 18 LDI $1B Nummer für 'file data'
AB68 4C 37 A4 JMP $A437 Fehlermeldung ausgeben
AB6B A9 0C LDA $0C
AB6D A0 AD LDY $AD Zeiger auf 'Predo from start'
AB6F 20 1E AB JSR $AB1E String ausgeben
AB72 A5 3D LDA $3D
AB74 A4 3E LDY $3E Programmzeiger zurück
AB76 B5 7A STA $7A
AB78 B4 7B STY $7B auf INPUT-Befehl
AB7A 60 RTS

***** BASIC-Befehl GET
AB7B 20 A6 B3 JSR $B3A6 Testet auf Direkt-Modus
AB7E C9 23 CMP $23 '0'
AB80 D0 10 BNE $AB92 nein ?
AB82 20 73 00 JSR $0073 CHRGET nächstes Zeichen holen
AB85 20 9E B7 JSR $B79E Byte-Wert holen
AB88 A9 2C LDA $2C ',' Komma
AB8A 20 FF AE JSR $AEFF prüft auf Kode
AB8D B6 13 STX $13
AB8F 20 1E E1 JSR $E11E CHKIN
AB92 A2 01 LDX $01
AB94 A0 02 LDY $02 Zeiger auf Pufferende = $201 ein Zeichen
AB96 A9 00 LDA $00 Puffer mit $0 abschließen
AB98 BD 01 02 STA $0201
AB9B A9 40 LDA $40 GET-Flag
AB9D 20 0F AC JSR $AC0F Wertzuweisung an Variable
ABA0 A6 13 LDX $13 Eingabegerät
ABA2 D0 13 BNE $ABB7 nicht Tastatur dann CLRCH
ABA4 60 RTS

***** BASIC-Befehl INPUT#
ABA5 20 9E B7 JSR $B79E holt Byte-Wert
ABAE A9 2C LDA $2C ','
ABAA 20 FF AE JSR $AEFF prüft auf Komma
ABAD B6 13 STX $13 Eingabegerät
ABAF 20 1E E1 JSR $E11E CHKIN
ABB2 20 CE AB JSR $ABCE INPUT ohne Dialogstring
ABB5 A5 13 LDA $13
ABB7 20 CC FF JSR $FFCC CLRCH setzt Eingabegerät zurück
ABBA A2 00 LDX $00
ABBC B6 13 STX $13 Eingabegerät wieder Tastatur
ABBE 60 RTS

***** BASIC-Befehl INPUT
ABBF C9 22 CMP $22 '=' Hochkomma
ABC1 D0 0B BNE $ABCE nein
ABC3 20 BD AE JSR $ABED Dialogstring holen
ABC6 A9 3B LDA $3B ';' Semikolon
ABC8 20 FF AE JSR $AEFF prüft auf Kode
ABCB 20 21 AB JSR $AB21 String ausgeben
ABCE 20 A6 B3 JSR $B3A6 prüft auf Direkt-Modus
ABD1 A9 2C LDA $2C ',' Komma
ABD3 BD FF 01 STA $01FF an Pufferstart
ABD6 20 F9 AB JSR $ABF9 Fragezeichen ausgeben
ABD9 A5 13 LDA $13 Nummer des Eingabegeräts
ABDB F0 0D BEQ $ABEA Tastatur ?

```

ABDD	20 B7 FF	JSR \$FFB7	Status holen
ABE0	29 02	AND #02	
ABE2	F0 06	BEO \$ABEA	Time-out ?
ABE4	20 B5 AB	JSR \$ABB5	ja, CLRCH, Tastatur wieder aktivieren
ABE7	4C FB AB	JMP \$ABFB	nächstes Statement ausführen

ABEA	AD 00 02	LDA \$0200	erstes Zeichen holen
ABED	D0 1E	BNE \$AC0D	Ende ?
ABEF	A5 13	LDA #13	ja, Eingabegerät
ABF1	D0 E3	BNE \$ABD6	nicht Tastatur ?
ABF3	20 06 A9	JSR \$A906	Offset des nächsten Statements suchen
ABF6	4C FB AB	JMP \$ABFB	Programmezeiger auf nächstes Statement

ABF9	A5 13	LDA #13	Eingabegerät
ABFB	D0 06	BNE \$AC03	nicht Tastatur ?
ABFD	20 45 AB	JSR \$AB45	'?' ausgeben
AC00	20 3B AB	JSR \$AB3B	' ' Leerzeichen ausgeben
AC03	4C 60 A5	JMP \$A560	Eingabezeile holen

\*\*\*\*\* BASIC-Befehl READ

AC06	A6 41	LDX \$41	
AC08	A4 42	LDY \$42	DATA-Zeiger holen
AC0A	A9 98	LDA #98	READ-Flag
AC0C	2C	.BYTE \$2C	
AC0D	A9 00	LDA #00	
AC0F	85 11	STA #11	setzen
AC11	86 43	STX \$43	
AC13	84 44	STY \$44	INPUT-Zeiger auf Eingabe
AC15	20 B8 B0	JSR \$B0BB	sucht Variable
AC18	B5 49	STA \$49	
AC1A	B4 4A	STY \$4A	Variablenadresse speichern
AC1C	A5 7A	LDA \$7A	
AC1E	A4 7B	LDY \$7B	
AC20	B5 4B	STA \$4B	Programmezeiger in \$4B/\$4C zwischenspeichern
AC22	B4 4C	STY \$4C	
AC24	A6 43	LDX \$43	
AC26	A4 44	LDY \$44	INPUT-Zeiger
AC28	86 7A	STX \$7A	gleich Programmezeiger
AC2A	B4 7B	STY \$7B	
AC2C	20 79 00	JSR \$0079	CHRGET letztes Zeichen holen
AC2F	D0 20	BNE \$AC51	
AC31	24 11	BIT #11	Eingabeflag
AC33	50 0C	BVC \$AC41	kein GET ?
AC35	20 24 E1	JSR \$E124	GETIN
AC38	BD 00 02	STA \$0200	Zeichen in Puffer schreiben
AC3B	A2 FF	LDX #FF	
AC3D	A0 01	LDY #01	
AC3F	D0 0C	BNE \$AC4D	
AC41	30 75	BMI \$ACBB	
AC43	A5 13	LDA #13	Eingabegerät
AC45	D0 03	BNE \$AC4A	nicht Tastatur ?
AC47	20 45 AB	JSR \$AB45	Fragezeichen ausgeben
AC4A	20 F9 AB	JSR \$ABF9	zweites Fragezeichen ausgeben
AC4D	B6 7A	STX \$7A	Programmezeiger setzen
AC4F	B4 7B	STY \$7B	
AC51	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
AC54	24 0D	BIT #0D	Typ-Flag
AC56	10 31	BPL \$ACB9	
AC58	24 11	BIT #11	Eingabeflag
AC5A	50 09	BVC \$AC65	kein GET ?

AC5C	EB	INX	
AC5D	86 7A	STX \$7A	
AC5F	A9 00	LDA #\$00	
AC61	85 07	STA \$07	
AC63	F0 0C	BEQ \$AC71	
AC65	85 07	STA \$07	
AC67	C9 22	CMP #\$22	''' Hochkomma
AC69	F0 07	BEQ \$AC72	
AC6B	A9 3A	LDA #\$3A	':'
AC6D	85 07	STA \$07	
AC6F	A9 2C	LDA #\$2C	','
AC71	1B	CLC	
AC72	85 0B	STA \$0B	
AC74	A5 7A	LDA \$7A	
AC76	A4 7B	LDY \$7B	
AC78	69 00	ADC #\$00	
AC7A	90 01	BCC \$AC7D	
AC7C	CB	INY	
AC7D	20 BD B4	JSR \$B48D	String übernehmen
AC80	20 E2 B7	JSR \$B7E2	Programmzeiger hinter String setzen
AC83	20 DA A9	JSR \$A9DA	String an Variable zuweisen
AC86	4C 91 AC	JMP \$AC91	weiter machen
AC89	20 F3 BC	JSR \$BCF3	Ziffernstring in FAC holen
AC8C	A5 0E	LDA \$0E	INTEGER/REAL-Flag
AC8E	20 C2 A9	JSR \$A9C2	FAC an numerische Variable zuweisen
AC91	20 79 00	JSR \$0079	CHRGOT letztes Zeichen holen
AC94	F0 07	BEQ \$AC9D	Ende ?
AC96	C9 2C	CMP #\$2C	','
AC98	F0 03	BEQ \$AC9D	
AC9A	4C 4D AB	JMP \$AB4D	zur Fehlerbehandlung
AC9D	A5 7A	LDA \$7A	
AC9F	A4 7B	LDY \$7B	Programmzeiger
ACA1	85 43	STA \$43	
ACA3	B4 44	STY \$44	gleich DATA-Zeiger
ACA5	A5 4B	LDA \$4B	
ACA7	A4 4C	LDY \$4C	Programmzeiger
ACA9	85 7A	STA \$7A	wieder zurückholen
ACAB	B4 7B	STY \$7B	
ACAD	20 79 00	JSR \$0079	CHRGOT letztes Zeichen holen
ACB0	F0 2D	BEQ \$ACDF	
ACB2	20 FD AE	JSR \$AEFD	CKCOM prüft auf Komma
ACB5	4C 15 AC	JMP \$AC15	weiter
ACB8	20 06 A9	JSR \$A906	nächstes Statement suchen
ACBB	CB	INY	
ACBC	AA	TAX	Zeilenende ?
ACBD	D0 12	BNE \$ACD1	nein
ACBF	A2 0D	LDX #\$0D	
ACC1	CB	INY	
ACC2	B1 7A	LDA (\$7A),Y	
ACC4	F0 6C	BEQ \$AD32	Programmende ? 'out of data' X = 0
ACC6	CB	INY	
ACC7	B1 7A	LDA (\$7A),Y	
ACC9	85 3F	STA \$3F	
ACCB	CB	INY	
ACCC	B1 7A	LDA (\$7A),Y	
ACCE	CB	INY	
ACCF	85 40	STA \$40	
ACD1	20 FB AB	JSR \$ABFB	Programmzeiger auf nächstes Statement
ACD4	20 79 00	JSR \$0079	CHRGOT letztes Zeichen holen
ACD7	AA	TAX	

ACDB	E0 83	CPX #83	'DATA'
ACDA	D0 DC	BNE \$ACBB	nein, weitersuchen
ACDC	4C 51 AC	JMP \$AC51	Daten lesen
ACDF	A5 43	LDA \$43	
ACE1	A4 44	LDY \$44	Input-Zeiger
ACE3	A6 11	LDX \$11	Eingabe-Flag
ACE5	10 03	BPL \$ACEA	kein DATA ?
ACE7	4C 27 AB	JMP \$AB27	DATA-Zeiger setzen
ACEA	A0 00	LDY #00	
ACEC	B1 43	LDA (\$43),Y	
ACEE	F0 0B	BEQ \$ACFB	
ACF0	A5 13	LDA \$13	
ACF2	D0 07	BNE \$ACFB	
ACF4	A9 FC	LDA #\$FC	
ACF6	A0 AC	LDY #AC	Zeiger auf '?extra ignored'
ACFB	4C 1E AB	JMP \$AB1E	String ausgeben
ACFB	60	RTS	
ACFC	3F 45 5B 54 52 41 20 49		'?extra ignored'
AD04	47 4E 4F 52 45 44 0B 00		
AD0C	3F 52 45 44 4F 20 46 52		'?redo from start'
AD14	4F 20 53 54 41 52 54 0D		
AD1C	00		
***** BASIC-Befehl NEXT			
AD1D	D0 04	BNE \$AD24	folgt Variablenname ?
AD20	A0 00	LDY #00	
AD22	F0 03	BEQ \$AD27	
AD24	20 8B B0	JSR \$B0BB	sucht Variable
AD27	B5 49	STA \$49	
AD29	B4 4A	STY \$4A	Variablenadresse
AD2B	20 BA A3	JSR \$A3BA	sucht FOR-NEXT-Schleife im Stack
AD2E	F0 05	BEQ \$AD35	gefunden
AD30	A2 0A	LDX #0A	Nummer für 'next without for'
AD32	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
AD35	9A	TXS	
AD36	8A	TXA	
AD37	1B	CLC	
AD38	69 04	ADC #04	
AD3A	4B	PHA	
AD3B	69 06	ADC #06	
AD3D	B5 24	STA \$24	
AD3F	6B	PLA	
AD40	A0 01	LDY #01	
AD42	20 A2 BB	JSR \$BBA2	Variable vom Stack in FAC holen
AD45	BA	TSX	
AD46	BD 09 01	LDA \$0109,X	
AD49	B5 66	STA \$66	
AD4B	A5 49	LDA \$49	Variablenadresse
AD4D	A4 4A	LDY \$4A	
AD4F	20 67 BB	JSR \$BB67	addiert STEP-Wert zu FAC
AD52	20 D0 BB	JSR \$BBD0	FAC nach Variable bringen
AD55	A0 01	LDY #01	
AD57	20 5D BC	JSR \$BC5D	FAC mit Schleifenendwert vergleichen
AD5A	BA	TSX	
AD5B	3B	SEC	
AD5C	FD 09 01	SBC \$0109,X	
AD5F	F0 17	BEQ \$AD7B	
AD61	BD 0F 01	LDA \$010F,X	
AD64	B5 39	STA \$39	

AD66	BD 10 01	LDA \$0110,X	Zeilennummer holen
AD69	85 3A	STA \$3A	
AD6B	BD 12 01	LDA \$0112,X	
AD6E	85 7A	STA \$7A	
AD70	BD 11 01	LDA \$0111,X	Programmzeiger holen
AD73	85 7B	STA \$7B	
AD75	4C AE A7	JMP \$A7AE	zur Interpreterschleife
AD78	8A	TXA	
AD79	69 11	ADC #\$11	
AD7B	AA	TAX	
AD7C	9A	TXS	
AD7D	20 79 00	JSR \$0079	CHRGOT letztes Zeichen holen
AD80	C9 2C	CMP #\$2C	',' Komma
AD82	D0 F1	BNE \$AD75	nein, dann fertig
AD84	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
AD87	20 24 AD	JSR \$AD24	nächste NEXT-Variable
***** FRMNUM Ausdruck holen und auf numerisch prüfen			
AD8A	20 9E AD	JSR \$AD9E	FRMEVL Term holen
***** prüft auf numerisch			
AD8D	1B	CLC	
ADBE	24	.BYTE \$24	
***** prüft auf String			
AD8F	3B	SEC	
AD90	24 0D	BIT \$0D	Typflag testen
AD92	30 03	BMI \$AD97	
AD94	B0 03	BCS \$AD99	
AD96	60	RTS	
AD97	B0 FD	BCS \$AD96	
AD99	A2 16	LDI #\$16	Nummer für 'type mismatch'
AD9B	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
***** FRMEVL auswerten eines beliebigen Ausdrucks			
AD9E	A6 7A	LDI \$7A	
ADA0	D0 02	BNE \$ADA4	Programmzeiger um eins erniedrigen
ADA2	C6 7B	DEC \$7B	
ADA4	C6 7A	DEC \$7A	
ADA6	A2 00	LDI #\$00	
ADAB	24	.BYTE \$24	
ADA9	4B	PHA	
ADAA	8A	TXA	
ADAB	4B	PHA	
ADAC	A9 01	LDA #\$01	2 Bytes
ADAE	20 FB A3	JSR \$A3FB	prüft auf genügend Platz im Stack
ADB1	20 B3 AE	JSR \$AEB3	Nächstes Element holen
ADB4	A9 00	LDA #\$00	
ADB6	B5 4D	STA \$4D	Maske für Vergleichsoperator
ADB8	20 79 00	JSR \$0079	CHRGOT letztes Zeichen holen
ADB9	3B	SEC	
ADBC	E9 B1	SBC #\$B1	
ADBE	90 17	BCC \$ADD7	
ADC0	C9 03	CMP #\$03	
ADC2	B0 13	BCS \$ADD7	
ADC4	C9 01	CMP #\$01	
ADC6	2A	ROL A	Maske für kleiner, gleich und größer
ADC7	49 01	EOR #\$01	
ADC9	43 4D	EOR \$4D	
ADCB	C5 4D	CMP \$4D	

ADCD	90 61	BCC \$AE30	
ADCF	85 4D	STA \$4D	
ADD1	20 73 00	JSR #0073	CHRGET nächstes Zeichen holen
ADD4	4C 8B AD	JMP \$ADBB	zurück
ADD7	A6 4D	LDX \$4D	
ADD9	D0 2C	BNE \$AE07	
ADDB	B0 7B	BCS \$AE58	
ADDD	69 07	ADC #07	
ADDF	90 77	BCC \$AE58	
ADE1	65 0D	ADC \$0D	
ADE3	D0 03	BNE \$ADE8	numerisch ?
ADE5	4C 3D B6	JMP \$B63D	Stringverkettung
ADE8	69 FF	ADC #FF	Kode - \$AA
ADEA	B5 22	STA \$22	
ADEC	0A	ASL A	
ADED	65 22	ADC \$22	mal 3
ADEF	AB	TAY	
ADF0	68	PLA	
ADF1	D9 B0 A0	CMP \$A0B0,Y	mit Hierarchieflag vergleichen
ADF4	B0 67	BCS \$AE5D	
ADF6	20 BD AD	JSR \$ADBD	prüft auf numerisch
ADF9	4B	PHA	
ADFA	20 20 AE	JSR \$AE20	Operatoradresse und Operanden auf Stack
ADFD	68	PLA	
ADFE	A4 4B	LDY \$4B	
AE00	10 17	BPL \$AE19	
AE02	AA	TAX	
AE03	F0 56	BEQ \$AE58	
AE05	D0 5F	BNE \$AE66	
AE07	46 DD	LSR \$0D	Stringflag löschen
AE09	BA	TXA	
AE0A	2A	ROL A	
AE0B	A6 7A	LDX \$7A	
AE0D	D0 02	BNE \$AE11	Programnzeiger um eins zurücksetzen
AE0F	C6 7B	DEC \$7B	
AE11	C6 7A	DEC \$7A	
AE13	A0 1B	LDY #1B	Offset des Hierarchieflags
AE15	B5 4D	STA \$4D	Flag setzen
AE17	D0 D7	BNE \$ADFO	
AE19	D9 B0 A0	CMP \$A0B0,Y	mit Hierarchieflag vergleichen
AE1C	B0 4B	BCS \$AE66	
AE1E	90 D9	BCC \$ADF9	
AE20	B9 B2 A0	LDA \$A0B2,Y	
AE23	4B	PHA	Operationsadresse auf Stack
AE24	B9 B1 A0	LDA \$A0B1,Y	
AE27	4B	PHA	
AE28	20 33 AE	JSR \$AE33	Operanden auf Stack
AE2B	A5 4D	LDA \$4D	
AE2D	4C A9 AD	JMP \$ADA9	zum Schleitenanfang
AE30	4C 0B AF	JMP \$AF0B	gibt 'syntax error'
AE33	A5 66	LDA \$66	Vorzeichen
AE35	BE B0 A0	LDX \$A0B0,Y	Hierarchieflag
AE38	AB	TAY	
AE39	68	PLA	
AE3A	B5 22	STA \$22	
AE3C	E6 22	INC \$22	Rucksprungadresse merken
AE3E	68	PLA	
AE3F	B5 23	STA \$23	
AE41	9B	TYA	Vorzeichen
AE42	4B	PHA	

AE43	20 1B BC	JSR \$8C1B	FAC runden
AE46	A5 65	LDA \$65	
AE48	4B	PHA	
AE49	A5 64	LDA \$64	
AE4B	4B	PHA	
AE4C	A5 63	LDA \$63	
AE4E	4B	PHA	FAC auf Stack
AE4F	A5 62	LDA \$62	
AE51	4B	PHA	
AE52	A5 61	LDA \$61	
AE54	4B	PHA	
AE55	6C 22 00	JMP (\$0022)	Sprung auf Operation
AE5B	A0 FF	LDY \$\$FF	
AE5A	6B	PLA	
AE5B	F0 23	BEQ \$AE80	
AE5D	C9 64	CMP \$64	
AE5F	F0 03	BEQ \$AE64	
AE61	20 3D AD	JSR \$ADBD	prüft auf numerisch
AE64	B4 4B	STY \$4B	
AE66	6B	PLA	
AE67	4A	LSR A	
AE68	B5 12	STA \$12	
AE6A	6B	PLA	
AE6B	B5 69	STA \$69	
AE6D	6B	PLA	
AE6E	B5 6A	STA \$6A	
AE70	6B	PLA	ARG vom Stack holen
AE71	B5 6B	STA \$6B	
AE73	6B	PLA	
AE74	B5 6C	STA \$6C	
AE76	6B	PLA	
AE77	B5 6D	STA \$6D	
AE79	6B	PLA	
AE7A	B5 6E	STA \$6E	
AE7C	45 66	EOR \$66	
AE7E	B5 6F	STA \$6F	
AE80	A5 61	LDA \$61	
AE82	60	RTS	
*****			
AE83	6C 0A 03	JMP (\$030A)	Nächstes Element eines Ausdrucks holen
AE86	A9 00	LDA \$00	JMP \$AE86
AE8B	B5 0D	STA \$0D	
AE8A	20 73 00	JSR \$0073	Typflag auf numerisch
AE8D	B0 03	BCS \$AE92	CHRGET nächstes Zeichen holen
AE8F	4C F3 BC	JMP \$BCF3	Ziffer ?
AE92	20 13 B1	JSR \$B113	Variable nach FAC holen
AE95	90 03	BCC \$AE9A	Buchstabe ?
AE97	4C 28 AF	JMP \$AF2B	nein
AE9A	C9 FF	CMP \$\$FF	Variable holen
AE9C	D0 0F	BNE \$AEAD	BASIC-Kode für Pi ?
AE9E	A9 4B	LDA \$4B	
AEA0	A0 AE	LDY \$AE	Zeiger auf Konstante Pi
AEA2	20 A2 BB	JSR \$BBA2	
AEA5	4C 73 00	JMP \$0073	Konstante in FAC holen
*****			
AEAB	B2 49 0F DA A1		CHRGET nachstes Zeichen holen
*****			
Konstante Pi 3.14159265			



A6AD	C9 2E	CMP #02E	'.'
A6AF	F0 DE	BEQ \$A6BF	
A6B1	C9 AB	CMP #0AB	'-'
A6B3	F0 5B	BEQ \$A6FD	zum Vorzeichenwechsel
A6B5	C9 AA	CMP #0AA	'+'
A6B7	F0 D1	BEQ \$A6BA	
A6B9	C9 22	CMP #022	''
A6BB	D0 0F	BNE \$A6CB	
A6BD	A5 7A	LDA \$7A	
A6BF	A4 7B	LDY \$7B	Programmzeiger holen
A6C1	69 00	ADC #00	
A6C3	90 01	BCC \$A6C6	
A6C5	C8	INY	
A6C6	20 B7 B4	JSR \$B4B7	String übertragen
A6C9	4C E2 B7	JMP \$B7E2	Programmzeiger auf Stringende + 1
A6CC	C9 AB	CMP #0AB	'NOT'-Kode
A6CE	D0 13	BNE \$A6E3	
A6D0	A0 1B	LDY #01B	Offset des Hierarchie-Flags in Tabelle
A6D2	D0 3B	BNE \$A6FF	
***** BASIC-Befehl NOT			
A6D4	20 BF B1	JSR \$B1BF	FAC nach INTEGER wandlen
A6D7	A5 65	LDA \$65	
A6D9	49 FF	EOR \$FF	alle Bits umdrehen
A6DB	AB	TAY	
A6DC	A5 64	LDA \$64	
A6DE	49 FF	EOR \$FF	
A6E0	4C 91 B3	JMP \$B391	wieder nach Fließkomma wandlen
*****			
A6E3	C9 A5	CMP #0A5	'FN'-Kode
A6E5	D0 03	BNE \$A6EA	
A6E7	4C F4 B3	JMP \$B3F4	FN ausführen
*****			
A6EA	C9 B4	CMP #0B4	'SGN'-Kode
A6EC	90 03	BCC \$A6F1	kleiner (keine Stringfunktion) ?
A6EE	4C A7 AF	JMP \$AFA7	holt String und ersten Parameter
*****			
A6F1	20 FA AE	JSR \$A6FA	holt Term in Klammern
A6F4	20 9E AD	JSR \$AD9E	prüft auf Klammer auf FRMEVL holt Term
*****			
A6F7	A9 29	LDA #029	prüft auf Zeichen in BASIC-Text
A6F9	2C	.BYTE \$2C	)' Klammer zu
A6FA	A9 2B	LDA #02B	
A6FC	2C	.BYTE \$2C	(' Klammer auf
A6FD	A9 2C	LDA #02C	
A6FF	A0 00	LDY #00	',' Komma
A701	D1 7A	CMP (\$7A),Y	mit laufendem Zeichen vergleichen
A703	D0 03	BNE \$A70B	keine Übereinstimmung ?
A705	4C 73 00	JMP \$0073	CHRGET nächstes Zeichen holen
A70B	A2 0B	LDX #00B	Nummer für 'syntax error'
A70A	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
*****			
A70D	A0 15	LDY #015	Offset Hierarchie-Kode für Vorzeichenwechsel
A70F	6B	PLA	
A710	6B	PLA	
A711	4C FA AD	JMP \$ADFA	zur Auswertung

```

***** prüft auf Variable innerhalb des BASICs
AF14 38 SEC
AF15 A5 64 LDA #64
AF17 E9 00 SBC #000 liegt Descriptor (#64/#65) zwischen
AF19 A5 65 LDA #65 $A000 und $E32A ?
AF1B E9 A0 SBC #A0
AF1D 90 08 BCC #AF27 wenn ja dann C=1
AF1F A9 A2 LDA #A2
AF21 E5 64 SBC #64
AF23 A9 E3 LDA #E3
AF25 E5 65 SBC #65
AF27 60 RTS

***** Variable holen
AF28 20 8B B0 JSR $B0BB Variable suchen
AF2B B5 64 STA #64
AF2D B4 65 STY #65 zeigt auf Variable bzw. Stringdescriptor
AF2F A6 45 LDX #45
AF31 A4 46 LDY #46 Variablenname
AF33 A5 0D LDA #0D Typflag
AF35 F0 26 BEQ #AF5D numerisch ?
AF37 A9 00 LDA #00
AF39 B5 70 STA #70
AF3B 20 14 AF JSR #AF14 Descriptor im Interpreter ?
AF3E 90 1C BCC #AF5C nein
AF40 E0 54 CPX #54 'T'
AF42 D0 18 BNE #AF5C
AF44 C0 C9 CPY #C9 'I$'
AF46 D0 14 BNE #AF5C
AF48 20 B4 AF JSR #AFB4 Zeit holen
AF4B B4 5E STY #5E
AF4D B8 DEY
AF4E B4 71 STY #71
AF50 A0 06 LDY #06 Länge 6 für TI$
AF52 B4 5D STY #5D
AF54 A0 24 LDY #24
AF56 20 B8 BE JSR #BE68 erzeugt String TI$
AF59 4C 6F B4 JMP #B46F
AF5C 60 RTS

AF5D 24 0E BIT #0E INTEGER/ REAL Flag
AF5F 10 0D BPL #AF6E REAL ?

***** Integervariable holen
AF61 A0 00 LDY #00
AF63 B1 64 LDA ($64),Y Integerzahl holen
AF65 AA TAX
AF66 CB INY
AF67 B1 64 LDA ($64),Y
AF69 AB TAY
AF6A BA TXA
AF6B 4C 91 B3 JMP #B391 und nach Fließkomma wandeln

***** REAL-Variable holen
AF6E 20 14 AF JSR #AF14 Descriptor im Interpreter ?
AF71 90 2D BCC #AFA0 nein
AF73 E0 54 CPX #54 'Y'
AF75 D0 18 BNE #AF92
AF77 C0 49 CPY #49 'I'
AF79 D0 25 BNE #AFA0

```

AF7B	20 B4 AF	JSR \$AFB4	TIME in FAC holen
AF7E	98	TYA	
AF7F	A2 A0	LDX `#\$A0	
AF81	4C 4F BC	JMP \$BC4F	
*****			Zeit holen
AF84	20 DE FF	JSR \$FFDE	TIME holen
AF87	B6 64	STX \$64	
AF89	84 63	STY \$63	
AF8B	B5 65	STA \$65	
AF8D	A0 00	LDY #00	und in Fließkommaakku
AF8F	B4 62	STY \$62	
AF91	60	RTS	
AF92	E0 53	CPX #53	'S'
AF94	D0 0A	BNE \$AFA0	
AF96	C0 54	CPY #54	'T'
AF98	D0 06	BNE \$AFA0	
AF9A	20 B7 FF	JSR \$FFB7	Status holen
AF9D	4C 3C BC	JMP \$BC3C	Byte in Akku in Fließkommaformat wandeln
*****			REAL-Variable holen
AFA0	A5 64	LDA \$64	
AFA2	A4 65	LDY \$65	Variablenadresse
AFA4	4C A2 BB	JMP \$BBA2	Variable in FAC holen
*****			Funktionsberechnung
AFA7	0A	ASL A	Funktionskode mal 2
AFA8	48	PHA	
AFA9	AA	TAX	
AFAA	20 73 00	JSR \$0073	CHRGET nächstes Zeichen
AFAE	E0 BF	CPX #BF	
AFAF	90 20	BCC \$AFD1	numerische Funktion ?
*****			Stringfunktion, String und ersten Parameter
AFB1	20 FA AE	JSR \$AEFA	prüft auf Klammer auf
AFB4	20 9E AD	JSR \$AD9E	FRHEVL holen beliebigen Term
AFB7	20 FD AE	JSR \$AEFD	prüft auf Komma
AFBA	20 BF AD	JSR \$ADBF	prüft auf String
AFBD	68	PLA	Funktionstoken für left\$, right\$, aid\$
AFBE	AA	TAX	
AFBF	A5 65	LDA \$65	
AFC1	48	PHA	Adresse des Stringdescriptors
AFC2	A5 64	LDA \$64	
AFC4	48	PHA	
AFC5	BA	TXA	
AFC6	48	PHA	Token auf den Stack
AFC7	20 9E B7	JSR \$B79E	holt Byte-Wert (2. Parameter der Funktion)
AFC8	68	PLA	Token zurückholen
AFCB	A8	TAY	
AFCF	BA	TXA	
AFFD	48	PHA	Bytewert auf Stack
AFFE	4C D6 AF	JMP \$AFD6	Routine ausführen
*****			numerische Funktion auswerten
AFD1	20 F1 AE	JSR \$AEF1	holt Term in Klammern (Funktionsargument)
AFD4	68	PLA	BASIC-Kode für Funktion
AFD5	A8	TAY	
AFD6	B9 EA 9F	LDA \$9FEA,Y	
AFD9	85 55	STA \$55	setzt Vektor für

AFDB	B9 EB 9F	LDA \$9FEB,Y	Funktionsberechnung
AFDE	B5 56	STA \$56	
AFE0	20 54 00	JSR \$0054	Funktion ausführen
AFE3	4C BD AD	JMP \$ADBD	prüft auf numerisch
*****			BASIC-Befehl OR
AFE6	A0 FF	LDY #\$FF	Flag für OR
AFEB	2C	.BYTE \$2C	
*****			BASIC-Befehl AND
AFE9	A0 00	LDY #\$00	Flag für AND
AFEB	B4 0B	STY \$0B	Flag setzen
AFED	20 BF B1	JSR \$B1BF	FAC nach INTEGER wandlen
AFF0	A5 64	LDA \$64	
AFF2	45 0B	EOR \$0B	
AFF4	B5 07	STA \$07	
AFF6	A5 65	LDA \$65	mit Flag verknüpfen und nach \$7/\$8
AFFB	45 0B	EOR \$0B	
AFFA	B5 0B	STA \$0B	
AFFC	20 FC BB	JSR \$BBFC	ARG nach FAC
AFFF	20 BF B1	JSR \$B1BF	FAC nach Integer
B002	A5 65	LDA \$65	
B004	45 0B	EOR \$0B	
B006	25 0B	AND \$0B	logische Verknüpfung
B008	45 0B	EOR \$0B	
B00A	AB	TAY	
B00B	A5 64	LDA \$64	
B00D	45 0B	EOR \$0B	
B00F	25 07	AND \$07	
B011	45 0B	EOR \$0B	
B013	4C 91 B3	JMP \$B391	wieder nach Fließkomma wandlen
*****			Vergleich
B016	20 90 AD	JSR \$AD90	prüft auf identischen Variablentyp
B019	B0 13	BCS \$B02E	String ? dann weiter
B01B	A5 6E	LDA \$6E	
B01D	09 7F	ORA #\$7F	ARG in Speicherformat
B01F	25 6A	AND \$6A	
B021	B5 6A	STA \$6A	
B023	A9 69	LDA #\$69	Adresse von ARG
B025	A0 00	LDY #\$00	
B027	20 5B BC	JSR \$BC5B	Vergleich ARG mit FAC
B02A	AA	TAX	
B02B	4C 61 B0	JMP \$B061	Ergebnis in FAC holen
*****			Stringvergleich
B02E	A9 00	LDA #\$00	
B030	B5 0D	STA \$0D	Stringflag löschen
B032	C6 4D	DEC \$4D	
B034	20 A6 B6	JSR \$B6A6	FRESTR
B037	B5 61	STA \$61	Stringlänge
B039	B6 62	STX \$62	
B03B	B4 63	STY \$63	Stringadresse
B03D	A5 6C	LDA \$6C	
B03F	A4 6D	LDY \$6D	Zeiger auf zweiten String
B041	20 AA B6	JSR \$B6AA	FRESTR
B044	B6 6C	STX \$6C	
B046	B4 6D	STY \$6D	Stringadresse des 2. Strings
B048	AA	TAX	
B049	3B	SEC	

B04A	E5 61	SBC #61	Langen vergleichen
B04C	F0 08	BEQ #B056	gleich ?
B04E	A9 01	LDA #01	
B050	90 04	BCC #B056	2. String kürzer
B052	A6 61	LDX #61	Länge des 1. Strings
B054	A9 FF	LDA #FFF	
B056	B5 66	STA #66	
B058	A0 FF	LDY #FFF	
B05A	E8	INX	
B05B	C8	INY	
B05C	CA	DEX	
B05D	D0 07	BNE #B066	
B05F	A6 66	LDX #66	
B061	30 0F	BMI #B072	
B063	18	CLC	
B064	90 0C	BCC #B072	
B066	B1 6C	LDA (#6C),Y	Vergleich der Strings
B068	D1 62	CMP (#62),Y	zeichenweise
B06A	F0 EF	BEQ #B05B	
B06C	A2 FF	LDX #FFF	
B06E	B0 02	BCS #B072	
B070	A2 01	LDX #01	
B072	E8	INX	
B073	BA	TXA	
B074	2A	ROL A	
B075	25 12	AND #12	
B077	F0 02	BEQ #B07B	
B079	A9 FF	LDA #FFF	
B07B	4C 3C BC	JMP #BC3C	Ergebnis nach FAC holen
B07E	20 FD AE	JSR #AEFD	CHKCOM prüft auf Komma
#####			BASIC-Befehl DIM
B081	AA	TAX	
B082	20 90 B0	JSR #B090	Variable dimensionieren
B085	20 79 00	JSR #0079	CHRGET letztes Zeichen holen
B088	D0 F4	BNE #B07E	nicht Ende, dann zur nächsten Variablen
B08A	60	RTS	
#####			Variable holen
B08B	A2 00	LDX #00	Flag für nicht dimensionieren
B08D	20 79 00	JSR #0079	CHRGET letztes Zeichen holen
B090	B6 0C	STX #0C	DIM-Flag setzen
B092	B5 45	STA #45	Variablenname
B094	20 79 00	JSR #0079	CHRGET letztes Zeichen holen
B097	20 13 B1	JSR #B113	prüft auf Buchstabe
B09A	B0 03	BCS #B09F	ja
B09C	4C 08 AF	JMP #AF08	'syntax error'
B09F	A2 00	LDX #00	
B0A1	B6 0D	STX #0D	Stringflag löschen
B0A3	B6 0E	STX #0E	Integerflag löschen
B0A5	20 73 00	JSR #0073	CHRGET nächstes Zeichen holen
B0A8	90 05	BCC #B0AF	Ziffer ?
B0AA	20 13 B1	JSR #B113	prüft auf Buchstabe
B0AD	90 0B	BCC #B0BA	nein
B0AF	AA	TAX	zweiter Buchstabe des Namens
B0B0	20 73 00	JSR #0073	CHRGET nächstes Zeichen holen
B0B3	90 FB	BCC #B0B0	Ziffer ?
B0B5	20 13 B1	JSR #B113	prüft auf Buchstabe
B0B8	B0 F6	BCS #B0B0	ja, weitere Zeichen überlesen

BOBA	C9 24	CMP #24	's'
BOBC	D0 06	BNE \$B0C4	nein
BOBE	A9 FF	LDA #FFF	
BOCO	85 0D	STA \$0D	Stringflag setzen
BOC2	D0 10	BNE \$B0D4	Sprung
BOC4	C9 25	CMP #25	'x'
BOC6	D0 13	BNE \$B0D6	nein
BOC8	A5 10	LDA \$10	Integer erlaubt ?
BOCA	D0 D0	BNE \$B09C	nein, 'syntax error'
BOCC	A9 80	LDA #80	
BOCE	85 0E	STA \$0E	Integerflag setzen
BOD0	05 45	ORA \$45	Bit 7 im Namen setzen
BOD2	85 45	STA \$45	
BOD4	8A	TXA	
BOD5	09 80	ORA #80	Zweiter Buchstabe des Namens
BOD7	AA	TAX	
BOD8	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
BODB	86 46	STX \$46	zweiten Buchstaben speichern
BODD	38	SEC	
BODE	05 10	ORA \$10	
BOE0	E9 28	SBC #28	'('
BOE2	D0 03	BNE \$B0E7	nicht Klammer auf ?
BOE4	4C D1 B1	JMP \$B1D1	dimensionierte Variable holen
BOE7	A0 00	LDY \$00	
BOE9	84 10	STY \$10	
BOEB	A5 2D	LDA \$2D	Zeiger auf Variablenanfang
BOED	A6 2E	LDX \$2E	
BOEF	86 60	STX \$60	
BOF1	85 5F	STA \$5F	zum Suchen merken
BOF3	E4 30	CPX \$30	
BOF5	D0 04	BNE \$B0FB	
BOF7	C5 2F	CMP \$2F	Ende der Variablen schon erreicht
BOF9	F0 22	BEQ \$B11D	ja, nicht gefunden, neu anlegen
BOFB	A5 45	LDA \$45	ersten Buchstabe des Namens
BOFD	D1 5F	CMP (\$5F),Y	mit Variablentabelle vergleichen
BOFF	D0 08	BNE \$B109	nein, weitersuchen
B101	A5 46	LDA \$46	zweiten Buchstaben
B103	C8	INY	
B104	D1 5F	CMP (\$5F),Y	vergleichen
B106	F0 7D	BEQ \$B185	
B108	88	DEY	
B109	18	CLC	
B10A	A5 5F	LDA \$5F	
B10C	69 07	ADC #07	Zeiger um 7 erhöhen (2 + 5 Byte REAL Variable)
B10E	90 E1	BCC \$B0F1	
B110	E8	INX	
B111	D0 DC	BNE \$B0EF	weiter suchen
***** prüft auf Buchstabe			
B113	C9 41	CMP #41	'A'
B115	90 05	BCC \$B11C	
B117	E9 5B	SBC #5B	'Z'-1
B119	38	SEC	ja, dann C = 1
B11A	E9 A5	SBC #A5	nein, dann C = 0
B11C	60	RTS	
***** Variable anlegen			
B11D	68	PLA	
B11E	48	PHA	Aufrufadresse prüfen
B11F	C9 2A	CMP #2A	Aufruf von FRMEVL ?

B121	D0 05	BNE \$B128	nein, dann neu anlegen
B123	A9 13	LDA #\$13	Zeiger auf Konstante 0
B125	A0 BF	LDY #\$BF	
B127	60	RTS	
B128	A5 45	LDA \$45	
B12A	A4 46	LDY \$46	Variablenname
B12C	C9 54	CMP #\$54	'T'
B12E	D0 08	BNE \$B13B	
B130	C0 C9	CPY #\$C9	'Is'
B132	F0 EF	BEQ \$B123	ja, TIs
B134	C0 49	CPY #\$49	'I'
B136	D0 03	BNE \$B13B	nein
B138	4C 0B AF	JMP \$AF0B	'syntax error'
B13B	C9 53	CMP #\$53	'S'
B13D	D0 04	BNE \$B143	
B13F	C0 54	CPY #\$54	'T'
B141	F0 F5	BEQ \$B13B	ST, dann 'syntax error'
B143	A5 2F	LDA \$2F	
B145	A4 30	LDY \$30	Zeiger auf Arraytabelle
B147	B5 5F	STA \$5F	
B149	B4 60	STY \$60	merken
B14B	A5 31	LDA \$31	
B14D	A4 32	LDY \$32	Zeiger auf Ende der Arraytabelle
B14F	B5 5A	STA \$5A	
B151	B4 5B	STY \$5B	merken
B153	18	CLC	
B154	69 07	ADC #\$07	um 7 verschieben für Anlage einer
B156	90 01	BCC \$B159	neuen Variablen
B158	C8	INY	
B159	B5 5B	STA \$5B	
B15B	B4 59	STY \$59	neues Blockende
B15D	20 B8 A3	JSR \$A3B8	Block verschieben
B160	A5 5B	LDA \$5B	
B162	A4 59	LDY \$59	
B164	C8	INY	
B165	B5 2F	STA \$2F	Zeiger auf Arraytabelle neu setzen
B167	B4 30	STY \$30	
B169	A0 00	LDY #\$00	
B16B	A5 45	LDA \$45	erster Buchstabe des Namens
B16D	91 5F	STA (\$5F),Y	
B16F	C8	INY	
B170	A5 46	LDA \$46	und zweiter Buchstabe speichern
B172	91 5F	STA (\$5F),Y	
B174	A9 00	LDA #\$00	
B176	C8	INY	
B177	91 5F	STA (\$5F),Y	
B179	C8	INY	
B17A	91 5F	STA (\$5F),Y	
B17C	C8	INY	5 mal null für Variablenwert
B17D	91 5F	STA (\$5F),Y	
B17F	C8	INY	
B180	91 5F	STA (\$5F),Y	
B182	C8	INY	
B183	91 5F	STA (\$5F),Y	
B185	A5 5F	LDA \$5F	
B187	18	CLC	
B188	69 02	ADC #\$02	zwei für Namen addieren
B18A	A4 60	LDY \$60	Zeiger auf Variable
B18C	90 01	BCC \$B18F	
B18E	C8	INY	

B18F	B5 47	STA #47	nach \$47/\$48
B191	B4 48	STY #48	
B193	60	RTS	

*****			berechnet Zeiger auf erstes Arrayelement
B194	A5 0B	LDA #0B	Anzahl der Dimensionen
B196	0A	ASL A	mal 2
B197	69 05	ADC #05	plus 5
B199	65 5F	ADC #5F	
B19B	A4 60	LDY #60	zu \$5F/\$60 addieren
B19D	90 01	BCC \$B1A0	
B19F	C8	INY	
B1A0	B5 5B	STA #5B	Ergebnis-Zeiger
B1A2	B4 59	STY #59	
B1A4	60	RTS	

*****			Konstante -32768
B1A5	90 80 00 00 00		

*****			Umwandlung FAC nach Integer
B1AA	20 BF B1	JSR \$B1BF	FAC nach Integer wandeln
B1AD	A5 64	LDA #64	Lo-Byte
B1AF	A4 65	LDY #65	Hi-Byte
B1B1	60	RTS	

*****			Ausdruck holen und nach Integer
B1B2	20 73 00	JSR #0073	CHRGET nachstes Zeichen holen
B1B5	20 9E AD	JSR \$AD9E	FRMEVL, Ausdruck auswerten
B1B8	20 BD AD	JSR \$ADBD	pruft auf numerisch
B1BB	A5 66	LDA #66	Vorzeichen ?
B1BD	30 0D	BMI \$B1CC	negativ, dann 'illegal quantity'
B1BF	A5 61	LDA #61	Exponent
B1C1	C9 90	CMF #90	Betrag größer 32768 ?
B1C3	90 09	BCC \$B1CE	nein
B1C5	A9 A5	LDA #A5	
B1C7	A0 B1	LDY #B1	Zeiger auf Konstante -32768
B1C9	20 5B BC	JSR \$BC5B	Vergleich FAC mit Konstante
B1CC	D0 7A	BNE \$B24B	ungleich, 'illegal quantity'
B1CE	4C 9B BC	JMP \$BC9B	wandelt Fließkomma in Integer

*****			dimensionierte Variable holen
B1D1	A5 0C	LDA #0C	DIM Flag
B1D3	05 0E	ORA #0E	Integer Flag
B1D5	4B	PHA	
B1D6	A5 0D	LDA #0D	String Flag
B1D8	4B	PHA	
B1D9	A0 00	LDY #00	Anzahl der Indizes
B1DB	9B	TYA	
B1DC	4B	PHA	
B1DD	A5 46	LDA #46	2. Buchstabe des Variablennamens
B1DF	4B	PHA	
B1E0	A5 45	LDA #45	erster Buchstabe des Variablennamens
B1E2	4B	PHA	
B1E3	20 B2 B1	JSR \$B1B2	Index holen und nach Integer
B1E6	6B	PLA	
B1E7	B5 45	STA #45	
B1E9	6B	PLA	Variablenname zurückholen
B1EA	B5 46	STA #46	
B1EC	6B	PLA	Anzahl der Indizes
B1ED	AB	TAY	



B1EE	BA	TSX	
B1EF	BD 02 01	LDA #0102,X	
B1F2	48	PHA	Flags vom Stapel holen
B1F3	BD 01 01	LDA #0101,X	
B1F6	48	PHA	
B1F7	A5 64	LDA #64	
B1F9	9D 02 01	STA #0102,X	
B1FC	A5 65	LDA #65	Index low und high auf Stapel
B1FE	9D 01 01	STA #0101,X	
B201	C8	INY	Anzahl der Indizes erhöhen
B202	20 79 00	JSR #0079	CHRGDT letztes Zeichen holen
B205	C9 2C	CMP #32C	'.' Komma ?
B207	F0 D2	BEQ #B1DB	ja, dann nächsten Index
B209	84 08	STY #08	Anzahl der Indizes speichern
B20B	20 F7 AE	JSR #AEF7	prüft auf Klammer zu
B20E	68	PLA	
B20F	B5 0D	STA #0D	
B211	68	PLA	Flags zurückholen
B212	B5 0E	STA #0E	
B214	29 7F	AND #7F	
B216	85 0C	STA #0C	
B218	A6 2F	LDX #2F	
B21A	A5 30	LDA #30	Zeiger auf Arraytabelle
B21C	86 5F	STX #5F	
B21E	B5 60	STA #60	Zeiger merken
B220	C5 32	CMP #32	
B222	D0 04	BNE #B228	
B224	E4 31	CPX #31	mit Tabellenende vergleichen
B226	F0 39	BEQ #B261	ja, nicht gefunden, dann anlegen
B228	A0 00	LDY #00	
B22A	B1 5F	LDA (#5F),Y	Namen aus Tabelle
B22C	C8	INY	
B22D	C5 45	CMP #45	mit gesuchtem Namen vergleichen
B22F	D0 06	BNE #B237	
B231	A5 46	LDA #46	
B233	D1 5F	CMP (#5F),Y	zweiter Buchstabe
B235	F0 16	BEQ #B24D	gefunden
B237	C8	INY	
B238	B1 5F	LDA (#5F),Y	
B23A	18	CLC	Feldlänge addieren
B23B	65 5F	ADC #5F	
B23D	AA	TAX	
B23E	C8	INY	
B23F	B1 5F	LDA (#5F),Y	
B241	65 60	ADC #60	
B243	90 D7	BCC #B21C	und weiter suchen
B245	A2 12	LDX #12	Nummer für 'bad subscript'
B247	2C	.BYTE #2C	
B248	A2 0E	LDX #0E	Nummer für 'illegal quantity'
B24A	4C 37 A4	JMP #A437	Fehlermeldung ausgeben
B24D	A2 13	LDX #13	Nummer für 'redim'd array'
B24F	A5 0C	LDA #0C	DIM-Flag null ?
B251	D0 F7	BNE #B24A	nein, dann Fehlermeldung
B253	20 94 B1	JSR #B194	setzt Zeiger auf erstes Arrayelement
B256	A5 08	LDA #08	Zahl der gefundenen Dimensionen
B258	A0 04	LDY #04	
B25A	D1 5F	CMP (#5F),Y	mit Dimensionen des Arrays vergleichen
B25C	D0 E7	BNE #B245	ungleich, dann 'bad subscript'

B25E	4C EA B2	JMP \$B2EA	sucht gewünschtes Arrayelement
*****			Arrayvariable anlegen
B261	20 94 B1	JSR \$B194	Länge des Arraykopfs
B264	20 08 A4	JSR \$A408	prüft auf genügend Platz im Speicher
B267	A0 00	LDY #00	
B269	B4 72	STY \$72	
B26B	A2 05	LDY #05	Defaultwert für Variablenlänge (REAL)
B26D	A5 45	LDA \$45	erster Buchstabe des Namens
B26F	91 5F	STA (\$5F),Y	in Arraytabelle
B271	10 01	BPL \$B274	kein Integer ?
B273	CA	DEX	
B274	C8	INY	
B275	A5 46	LDA \$46	zweiter Buchstabe
B277	91 5F	STA (\$5F),Y	in Tabelle schreiben
B279	10 02	BPL \$B27D	kein String oder Integer ?
B27B	CA	DEX	
B27C	CA	DEX	
B27D	B6 71	STX \$71	endgültige Variablenlänge 2, 3 oder 5
B27F	A5 0B	LDA \$0B	Anzahl der Dimensionen
B281	C8	INY	
B282	C8	INY	
B283	C8	INY	
B284	91 5F	STA (\$5F),Y	speichern
B286	A2 0B	LDX #0B	11. Defaultwert für Dimensionierung
B288	A9 00	LDA #00	
B28A	24 0C	BIT \$0C	Aufruf durch DIM-Befehl ?
B28C	50 0B	BVC \$B296	nein
B28E	6B	PLA	Dimension vom Stack holen
B28F	1B	CLC	
B290	69 01	ADC #01	eins addieren
B292	AA	TAX	
B293	6B	PLA	
B294	69 00	ADC #00	
B296	C8	INY	
B297	91 5F	STA (\$5F),Y	und speichern
B299	C8	INY	
B29A	BA	TXA	
B29B	91 5F	STA (\$5F),Y	
B29D	20 4C B3	JSR \$B34C	Platz für weitere Dimensionen berechnen
B2A0	B6 71	STX \$71	
B2A2	B5 72	STA \$72	Variablenende-Zeiger merken
B2A4	A4 22	LDY \$22	
B2A6	C6 0B	DEC \$0B	weitere Dimensionen ?
B2AB	D0 DC	BNE \$B2B6	ja
B2AA	65 59	ADC \$59	
B2AC	B0 5D	BCS \$B30B	Feldlänge plus Startadresse
B2AE	B5 59	STA \$59	
B2B0	AB	TAY	
B2B1	BA	TXA	
B2B2	65 5B	ADC \$5B	
B2B4	90 03	BCC \$B2B9	
B2B6	C8	INY	
B2B7	F0 52	BEQ \$B30B	
B2B9	20 0B A4	JSR \$A40B	prüft auf genügend Speicherplatz
B2BC	B5 31	STA \$31	
B2BE	B4 32	STY \$32	Zeiger auf Ende der Arraytabelle
B2C0	A9 00	LDA #00	Array mit Nullen füllen
B2C2	E6 72	INC \$72	
B2C4	A4 71	LDY \$71	

B2C6	F0 05	BEQ \$B2CD	
B2C8	88	DEY	
B2C9	91 58	STA (\$58),Y	
B2CB	D0 FB	BNE \$B2CB	
B2CD	C6 59	DEC \$59	
B2CF	C6 72	DEC \$72	
B2D1	D0 F5	BNE \$B2CB	
B2D3	E6 59	INC \$59	
B2D5	38	SEC	
B2D6	A5 31	LDA \$31	
B2D8	E5 5F	SBC \$5F	
B2DA	A0 02	LDY #02	
B2DC	91 5F	STA (\$5F),Y	Arraylänge low
B2DE	A5 32	LDA \$32	
B2E0	C8	INY	
B2E1	E5 60	SBC \$60	
B2E3	91 5F	STA (\$5F),Y	Arraylänge high
B2E5	A5 0C	LDA \$0C	Aufruf von DIM-Befehl ?
B2E7	D0 62	BNE \$B34B	ja, RTS

\*\*\*\*\* Arrayelement suchen

B2E9	C8	INY	
B2EA	B1 5F	LDA (\$5F),Y	Zahl der Dimensionen
B2EC	85 08	STA \$08	speichern
B2EE	A9 00	LDA #00	
B2F0	85 71	STA \$71	
B2F2	85 72	STA \$72	
B2F4	C8	INY	
B2F5	68	PLA	
B2F6	AA	TAX	
B2F7	85 64	STA \$64	Index vom Stapel holen
B2F9	68	PLA	
B2FA	85 65	STA \$65	
B2FC	D1 5F	CMP (\$5F),Y	mit Wert in Array vergleichen
B2FE	90 0E	BCC \$B30E	kleiner ?
B300	D0 06	BNE \$B30B	größer, dann 'bad subscript'
B302	C8	INY	
B303	BA	TXA	
B304	D1 5F	CMP (\$5F),Y	bei Gleichheit low Byte vergleichen
B306	90 07	BCC \$B30F	kleiner, dann weiter
B308	4C 45 B2	JMP \$B245	'bad subscript'
B30B	4C 35 A4	JMP \$A435	'out of memory'

\*\*\*\*\* Berechnung der Adresse eines Arrayelements

B30E	C8	INY	
B30F	A5 72	LDA \$72	
B311	05 71	ORA \$71	
B313	18	CLC	
B314	F0 0A	BEQ \$B320	
B316	20 4C B3	JSR \$B34C	Multiplikation
B319	BA	TXA	
B31A	65 64	ADC \$64	
B31C	AA	TAX	
B31D	98	TYA	
B31E	A4 22	LDY \$22	
B320	65 65	ADC \$65	
B322	86 71	STX \$71	
B324	C6 08	DEC \$08	Anzahl der Dimensionen
B326	D0 CA	BNE \$B2F2	mit nächsten Index weitermachen
B328	85 72	STA \$72	

B32A	A2 05	LDX #005	Default für Variablenlänge (5, REAL)
B32C	A5 45	LDA #45	erster Buchstabe des Namens
B32E	10 01	BPL #B331	
B330	CA	DEX	
B331	A5 46	LDA #46	zweiter Buchstabe des Namens
B333	10 02	BPL #B337	
B335	CA	DEX	
B336	CA	DEX	
B337	86 28	STX #28	Länge der Variablen 2, 3 oder 5
B339	A9 00	LDA #000	
B33B	20 55 B3	JSR #B355	Offset in Array berechnen
B33E	BA	TXA	
B33F	65 58	ADC #58	
B341	85 47	STA #47	
B343	98	TYA	
B344	65 59	ADC #59	
B346	85 48	STA #48	
B348	A8	TAY	
B349	A5 47	LDA #47	
B34B	60	RTS	

\*\*\*\*\* Hilfsroutine für Arrayberechnung

B34C	B4 22	STY #22	
B34E	B1 5F	LDA (\$F),Y	
B350	85 28	STA #28	
B352	88	DEY	
B353	B1 5F	LDA (\$F),Y	
B355	85 29	STA #29	
B357	A9 10	LDA #10	
B359	85 5D	STA #5D	
B35B	A2 00	LDX #000	
B35D	A0 00	LDY #000	
B35F	BA	TXA	
B360	0A	ASL A	
B361	AA	TAX	
B362	98	TYA	
B363	2A	ROL A	
B364	A8	TAY	
B365	B0 A4	BCS #B30B	'out of memory'
B367	06 71	ASL #71	
B369	26 72	ROL #72	
B36B	90 0B	BCC #B37B	
B36D	18	CLC	
B36E	BA	TXA	
B36F	65 28	ADC #28	
B371	AA	TAX	
B372	98	TYA	
B373	65 29	ADC #29	
B375	A8	TAY	
B376	B0 93	BCS #B30B	'out of memory'
B378	C6 5D	DEC #5D	
B37A	D0 E3	BNE #B35F	
B37C	60	RTS	

\*\*\*\*\* BASIC-Funktion FRE

B37D	A5 0D	LDA #0D	Typflag
B37F	F0 03	BEQ #B384	kein String
B381	20 A6 B6	JSR #B6A6	FRESTR
B384	20 26 B5	JSR #B526	Garbage Collection
B387	38	SEC	

B38B	A5 33	LDA \$33	
B38A	E5 31	SBC \$31	Stringanfang
B38C	A8	TAY	
B38D	A5 34	LDA \$34	
B38F	E5 32	SBC \$32	minus Variablenende
B391	A2 00	LDX #000	
B393	B6 0D	STX \$0D	Flag auf numerisch setzen
B395	B5 62	STA \$62	
B397	B4 63	STY \$63	Ergebnis merken
B399	A2 90	LDX #090	
B39B	4C 44 BC	JMP \$BC44	und nach Fließkomma wandeln
***** BASIC-Funktion POS			
B39E	38	SEC	C=1 Cursorposition holen
B39F	20 F0 FF	JSR \$FFF0	Cursorposition holen
B3A2	A9 00	LDA #000	
B3A4	F0 EB	BEQ \$B391	weiter wie oben
***** Test auf Direkt-Modus			
B3A6	A6 3A	LDX \$3A	
B3A8	EB	INX	
B3A9	D0 A0	BNE \$B34B	nein, dann RTS
B3AB	A2 15	LDX #15	Nummer für 'illegal direct'
B3AD	2C	.BYTE \$2C	
B3AE	A2 1B	LDX #1B	Nummer für 'undef'd function'
B3B0	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
***** BASIC-Befehl DEF FN			
B3B3	20 E1 B3	JSR \$B3E1	prüft FN-Syntax
B3B6	20 A6 B3	JSR \$B3A6	testet auf Direkt-Modus
B3B9	20 FA AE	JSR \$AEFA	prüft auf 'Klammer auf'
B3BC	A9 80	LDA #80	
B3BE	B5 10	STA \$10	sperrt INTEGER-Variable
B3C0	20 BB B0	JSR \$B0BB	sucht Variable
B3C3	20 BD AD	JSR \$ADBD	prüft auf numerisch
B3C6	20 F7 AE	JSR \$AEF7	prüft auf 'Klammer zu'
B3C9	A9 B2	LDA #B2	'=' BASIC-Kode
B3CB	20 FF AE	JSR \$AEFF	prüft auf '='
B3CE	4B	PHA	
B3CF	A5 4B	LDA \$4B	
B3D1	4B	PHA	FN-Variablen-Adresse auf Stack
B3D2	A5 47	LDA \$47	
B3D4	4B	PHA	
B3D5	A5 7B	LDA \$7B	
B3D7	4B	PHA	Programmzeiger auf Stack
B3D8	A5 7A	LDA \$7A	
B3DA	4B	PHA	
B3DB	20 F8 AB	JSR \$ABF8	Programmzeiger auf nächstes Statement
B3DE	4C 4F B4	JMP \$B44F	FN-Variable vom Stack holen
***** prüft FN-Syntax			
B3E1	A9 A5	LDA #A5	FN-Kode
B3E3	20 FF AE	JSR \$AEFF	prüft auf FN-Kode
B3E6	09 B0	ORA #B0	
B3E8	B5 10	STA \$10	sperrt INTEGER-Variable
B3EA	20 92 B0	JSR \$B092	sucht Variable
B3ED	B5 4E	STA \$4E	
B3EF	B4 4F	STY \$4F	FN-Variablenzeiger setzen
B3F1	4C BD AD	JMP \$ADBD	prüft auf numerisch

```

***** BASIC-Funktion FN
B3F4 20 E1 B3 JSR #B3E1 prüft FN-Syntax
B3F7 A5 4F LDA $4F
B3F9 48 PHA FN-Variablenzeiger auf Stack
B3FA A5 4E LDA $4E
B3FC 48 PHA
B3FD 20 F1 AE JSR #AEF1 holt Term in Klammern
B400 20 BD AD JSR #ADBD prüft auf numerisch
B403 48 PLA
B404 05 4E STA $4E
B406 48 PLA FN-Variablenzeiger zuruckholen
B407 05 4F STA $4F
B409 A0 02 LDY #02
B40B B1 4E LDA ($4E),Y
B40D 05 47 STA $47
B40F AA TAX
B410 C8 INY
B411 B1 4E LDA ($4E),Y
B413 F0 99 BEQ #B3AE gibt 'undef'd function'
B415 05 48 STA $48
B417 C8 INY
B418 B1 47 LDA ($47),Y
B41A 48 PHA
B41B 08 DEY
B41C 10 FA BPL #B41B
B41E A4 48 LDY $48
B420 20 D4 BD JSR #BBD4 FAC in FN-Variable übertragen
B423 A5 7B LDA $7B
B425 48 PHA
B426 A5 7A LDA $7A
B428 48 PHA
B429 B1 4E LDA ($4E),Y
B42B 05 7A STA $7A
B42D C8 INY Programmzeiger auf FN-Ausdruck
B42E B1 4E LDA ($4E),Y
B430 05 7B STA $7B
B432 A5 4B LDA $4B
B434 48 PHA
B435 A5 47 LDA $47
B437 48 PHA
B438 20 BA AD JSR #ADBA FRMNUM numerischen Ausdruck holen
B43B 68 PLA
B43C 05 4E STA $4E
B43E 68 PLA
B43F 05 4F STA $4F
B441 20 79 00 JSR #0079 CHR60T letztes Zeichen holen
B444 F0 03 BEQ #B449 keine weitere Zeichen ?
B446 4C 08 AF JMP #AF08 gibt 'syntax error'

B449 68 PLA
B44A 05 7A STA $7A
B44C 68 PLA Programmzeiger
B44D 05 7B STA $7B
B44F A0 00 LDY #00
B451 68 PLA
B452 91 4E STA ($4E),Y
B454 68 PLA
B455 C8 INY
B456 91 4E STA ($4E),Y
B458 68 PLA

```

B459	C8	INY	und FN-Variable vom Stack holen
B45A	91 4E	STA (\$4E),Y	
B45C	68	PLA	
B45D	C8	INY	
B45E	91 4E	STA (\$4E),Y	
B460	68	PLA	
B461	C8	INY	
B462	91 4E	STA (\$4E),Y	
B464	60	RTS	
***** BASIC-Funktion STR\$			
B465	20 BD AD	JSR \$ADBD	prüft auf numerisch
B468	A0 00	LDY #00	
B46A	20 DF BD	JSR \$BDDF	FAC nach ASCII umwandeln
B46D	68	PLA	
B46E	68	PLA	
B46F	A9 FF	LDA #FF	
B471	A0 00	LDY #00	Startadresse des Strings = #FF
B473	F0 12	BEQ \$B4B7	
***** Stringzeiger berechnen			
B475	A6 64	LDX #64	Akku enthält Stringlänge
B477	A4 65	LDY #65	
B479	B6 50	STX #50	Zeiger auf Stringdescriptor
B47B	B4 51	STY #51	
B47D	20 F4 B4	JSR \$B4F4	schafft Platz für neuen String, Länge in A
B480	B6 62	STX #62	Adresse low
B482	B4 63	STY #63	Adresse high
B484	B5 61	STA #61	Länge
B486	60	RTS	
***** String holen, Zeiger in A/Y			
B487	A2 22	LDX #22	..
B489	B6 07	STX #07	
B48B	B6 08	STX #08	
B48D	B5 6F	STA #6F	Startadresse des Strings
B48F	B4 70	STY #70	
B491	B5 62	STA #62	
B493	B4 63	STY #63	
B495	A0 FF	LDY #FF	
B497	C8	INY	Zeiger erhöhen
B498	B1 6F	LDA (\$6F),Y	nächstes Zeichen des Strings
B49A	F0 0C	BEQ \$B4A8	Endekennzeichen ?
B49C	C5 07	CMR #07	
B49E	F0 04	BEQ \$B4A4	
B4A0	C5 08	CMR #08	
B4A2	D0 F3	BNE \$B497	
B4A4	C9 22	CMR #22	..
B4A6	F0 01	BEQ \$B4A9	
B4A8	18	CLC	
B4A9	B4 61	STY #61	Länge des Strings
B4AB	98	TYA	
B4AC	65 6F	ADC #6F	
B4AE	B5 71	STA #71	Endadresse low + 1
B4B0	A6 70	LDX #70	
B4B2	90 01	BCC \$B4B5	
B4B4	E8	INX	
B4B5	B6 72	STX #72	Endadresse high + 1
B4B7	A5 70	LDA #70	Startadresse high
B4B9	F0 04	BEQ \$B4BF	null ?

B4BB	C9 02	CMP #02	zwei ?
B4BD	D0 08	BNE \$B4CA	nein
B4BF	9B	TYA	Länge in Akku
B4C0	20 75 B4	JSR \$B475	Stringzeiger berechnen
B4C3	A6 6F	LDI #6F	
B4C5	A4 70	LDY #70	Startadresse holen
B4C7	20 8B B6	JSR \$B68B	String in Stringbereich kopieren
*****			Stringzeiger in Descriptorstack bringen
B4CA	A6 16	LDX #16	Stringdescriptor-Zeiger
B4CC	E0 22	CPX #22	Stringstack voll ?
B4CE	D0 05	BNE \$B4D5	nein
B4D0	A2 19	LDX #19	Nummer für 'formula too complex'
B4D2	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
B4D5	A5 61	LDA #61	
B4D7	95 00	STA #00,X	Stringlänge
B4D9	A5 62	LDA #62	
B4DB	95 01	STA #01,X	und Adresse in
B4DD	A5 63	LDA #63	
B4DF	95 02	STA #02,X	Stringstack bringen
B4E1	A0 00	LDY #00	
B4E3	B6 64	STI #64	
B4E5	B4 65	STY #65	Zeiger jetzt auf Descriptor
B4E7	B4 70	STY #70	
B4E9	8B	DEY	
B4EA	B4 0D	STY #0D	Stringflag setzen #FF
B4EC	86 17	STI #17	Index des letzten Stringdescriptors
B4EE	E8	INX	
B4EF	E8	INX	um drei erhöhen
B4F0	E8	INX	
B4F1	B6 16	STX #16	als neuen Index merken
B4F3	60	RTS	
*****			Platz für String reservieren, Länge in A
B4F4	46 0F	LSR #0F	Flag für Garbage Collect rucksetzen
B4F6	4B	PHA	Stringlänge
B4F7	49 FF	EOR #FF	
B4F9	3B	SEC	
B4FA	65 33	ADC #33	
B4FC	A4 34	LDY #34	
B4FE	B0 01	BCC \$B501	
B500	8B	DEY	
B501	C4 32	CPY #32	
B503	90 11	BCC \$B516	zuwenig Platz, dann Garbage Collect durchführen
B505	D0 04	BNE \$B50B	
B507	C5 31	CMP #31	
B509	90 0B	BCC \$B516	
B50B	B5 33	STA #33	
B50D	B4 34	STY #34	
B50F	B5 35	STA #35	
B511	B4 36	STY #36	
B513	AA	TAX	
B514	6B	PLA	Stringlänge zurückholen
B515	60	RTS	
*****			
B516	A2 10	LDX #10	Nummer für 'out of memory'
B518	A5 0F	LDA #0F	Flag für Garbage Collect
B51A	30 B6	BMI \$B4D2	bereits durchgeführt, dann 'out of memory'
B51C	20 26 B5	JSR \$B526	Garbage Collect
B51F	A9 B0	LDA #B0	Flag setzen



B521	B5 0F	STA \$0F	
B523	68	PLA	Stringlänge
B524	D0 D0	BNE \$B4F6	

*****			Garbage Collection
			ungültige Strings beseitigen
B526	A6 37	LDX \$37	
B528	A5 38	LDA \$38	
B52A	B6 33	STX \$33	
B52C	B5 34	STA \$34	
B52E	A0 00	LDY #00	
B530	B4 4F	STY \$4F	
B532	B4 4E	STY \$4E	
B534	A5 31	LDA \$31	
B536	A6 32	LDX \$32	
B538	B5 5F	STA \$5F	
B53A	B6 60	STX \$60	
B53C	A9 19	LDA #19	
B53E	A2 00	LDX #00	
B540	B5 22	STA \$22	
B542	B6 23	STX \$23	
B544	C5 16	CMP \$16	
B546	F0 05	BEQ \$B54D	
B548	20 C7 B5	JSR \$B5C7	
B54B	F0 F7	BEQ \$B544	
B54D	A9 07	LDA #07	
B54F	B5 53	STA \$53	
B551	A5 2D	LDA \$2D	
B553	A6 2E	LDX \$2E	
B555	B5 22	STA \$22	
B557	B6 23	STX \$23	
B559	E4 30	CPX \$30	
B55B	D0 04	BNE \$B561	
B55D	C5 2F	CMP \$2F	
B55F	F0 05	BEQ \$B566	
B561	20 BD B5	JSR \$B5BD	
B564	F0 F3	BEQ \$B559	
B566	B5 58	STA \$58	
B568	B6 59	STX \$59	
B56A	A9 03	LDA #03	
B56C	B5 53	STA \$53	
B56E	A5 58	LDA \$58	
B570	A6 59	LDX \$59	
B572	E4 32	CPX \$32	
B574	D0 07	BNE \$B57D	
B576	C5 31	CMP \$31	
B578	D0 03	BNE \$B57D	
B57A	4C 06 B6	JMP \$B606	
B57D	B5 22	STA \$22	
B57F	B6 23	STX \$23	
B581	A0 00	LDY #00	
B583	B1 22	LDA (\$22),Y	
B585	AA	TAX	
B586	C8	INY	
B587	B1 22	LDA (\$22),Y	
B589	08	PHP	
B58A	C8	INY	
B58B	B1 22	LDA (\$22),Y	
B58D	65 58	ADC \$58	
B58F	B5 58	STA \$58	
B591	C8	INY	

B592	B1 22	LDA (\$22),Y
B594	65 59	ADC #59
B596	85 59	STA #59
B598	28	PLP
B599	10 03	BPL #B56E
B59B	8A	TXA
B59C	30 00	BMI #B56E
B59E	C8	INY
B59F	B1 22	LDA (\$22),Y
B5A1	A0 00	LDY #00
B5A3	0A	ASL A
B5A4	69 05	ADC #05
B5A6	65 22	ADC #22
B5A8	85 22	STA #22
B5AA	90 02	BCC #B5AE
B5AC	E6 23	INC #23
B5AE	A6 23	LDX #23
B5B0	E4 59	CPX #59
B5B2	D0 04	BNE #B5BB
B5B4	C5 58	CMP #58
B5B6	F0 BA	BEQ #B572
B5B8	20 C7 B5	JSR #B5C7
B5BB	F0 F3	BEQ #B5B0

\*\*\*\*\* prüft auf Beseitigungsmöglichkeit

B5BD	B1 22	LDA (\$22),Y
B5BF	30 35	BMI #B5F6
B5C1	C8	INY
B5C2	B1 22	LDA (\$22),Y
B5C4	10 30	BPL #B5F6
B5C6	C8	INY
B5C7	B1 22	LDA (\$22),Y
B5C9	F0 28	BEQ #B5F6
B5CB	C8	INY
B5CC	B1 22	LDA (\$22),Y
B5CE	AA	TAX
B5CF	C8	INY
B5D0	B1 22	LDA (\$22),Y
B5D2	C5 34	CMP #34
B5D4	90 06	BCC #B5DC
B5D6	D0 1E	BNE #B5F6
B5D8	E4 33	CPX #33
B5DA	B0 1A	BCS #B5F6
B5DC	C5 60	CMP #60
B5DE	90 16	BCC #B5F6
B5E0	D0 04	BNE #B5E6
B5E2	E4 5F	CPX #5F
B5E4	90 10	BCC #B5F6
B5E6	86 5F	STX #5F
B5E8	85 60	STA #60
B5EA	A5 22	LDA #22
B5EC	A6 23	LDX #23
B5EE	85 4E	STA #4E
B5F0	86 4F	STX #4F
B5F2	A5 53	LDA #53
B5F4	85 55	STA #55
B5F6	A5 53	LDA #53
B5F8	18	CLC
B5F9	65 22	ADC #22
B5FB	85 22	STA #22

```

B5FD 90 02      BCC $B601
B5FF E6 23      INC #23
B601 A6 23      LDX #23
B603 A0 00      LDY #00
B605 60         RTS

```

#### \*\*\*\*\* Strings zusammenfügen

```

B606 A5 4F      LDA $4F
B608 05 4E      ORA $4E
B60A F0 F5      BEQ $B601
B60C A5 55      LDA $55
B60E 29 04      AND #04
B610 4A         LSR A
B611 AB         TAY
B612 85 55      STA $55
B614 B1 4E      LDA ($4E),Y
B616 65 5F      ADC $5F      $5F/$60 = Zeiger auf alten Blockanfang
B618 85 5A      STA $5A
B61A A5 60      LDA $60
B61C 69 00      ADC #00
B61E 85 5B      STA $5B      $5A/$5B = Zeiger auf altes Blockende
B620 A5 33      LDA $33
B622 A6 34      LDX $34
B624 85 5B      STA $5B      $5B/$59 = Zeiger auf neues Blockende
B626 86 59      STX $59
B628 20 BF A3   JSR $A3BF      Strings verschieben
B62B A4 55      LDY $55
B62D C8         INY
B62E A5 5B      LDA $5B
B630 91 4E      STA ($4E),Y
B632 AA         TAX
B633 E6 59      INC #59
B635 A5 59      LDA $59
B637 C8         INY
B638 91 4E      STA ($4E),Y
B63A 4C 2A B5   JMP $B52A

```

#### \*\*\*\*\* Stringverknüpfung '+'

```

B63D A5 65      LDA $65
B63F 4B         PHA           Descriptor des ersten String merken
B640 A5 64      LDA $64
B642 4B         PHA
B643 20 B3 AE   JSR $AEB3      zweiten String holen
B646 20 BF AD   JSR $ADBF      prüft auf Stringvariable
B649 68         PLA
B64A 85 6F      STA $6F
B64C 68         PLA           Descriptor zurückholen
B64D 85 70      STA $70
B64F A0 00      LDY #00
B651 B1 6F      LDA ($6F),Y      Länge des ersten Strings
B653 1B         CLC
B654 71 64      ADC ($64),Y      plus Länge des zweiten Strings
B656 90 05      BCC $B65D      kleiner als 256
B658 A2 17      LDX #17         Nummer für 'string too long'
B65A 4C 37 A4   JMP $A437      Fehlermeldung ausgeben

B65D 20 75 B4   JSR $B475      Platz für verknüpften String reservieren
B660 20 7A B6   JSR $B67A      ersten String dorthin übertragen
B663 A5 50      LDA $50
B665 A4 51      LDY $51         Zeiger auf zweiten Stringdescriptor

```

B667	20 AA B6	JSR \$B6AA	FRESTR
B66A	20 8C B6	JSR \$B68C	zweiten String an ersten anhängen
B66D	A5 6F	LDA \$6F	
B66F	A4 70	LDY \$70	
B671	20 AA B6	JSR \$B6AA	FRESTR
B674	20 CA B4	JSR \$B4CA	Descriptor in Stringstack
B677	4C 8B AD	JMP \$ADB8	zurück zur Formelauswertung
***** String in reservierten Bereich übertragen			
B67A	A0 00	LDY \$00	
B67C	B1 6F	LDA (\$6F),Y	Stringlänge merken
B67E	4B	PHA	
B67F	CB	INY	
B680	B1 6F	LDA (\$6F),Y	Stringadresse low
B682	AA	TAX	
B683	CB	INY	
B684	B1 6F	LDA (\$6F),Y	Stringadresse high
B686	AB	TAY	
B687	AB	PLA	Stringlänge
B688	86 22	STX \$22	
B68A	84 23	STY \$23	Zeiger auf String
B68C	AB	TAY	
B68D	F0 0A	BEQ \$B699	Länge null, dann fertig
B68F	4B	PHA	
B690	8B	DEY	
B691	B1 22	LDA (\$22),Y	String übertragen
B693	91 35	STA (\$35),Y	in Stringbereich übertragen
B695	9B	TYA	
B696	D0 FB	BNE \$B690	
B698	6B	PLA	
B699	1B	CLC	
B69A	65 35	ADC \$35	
B69C	85 35	STA \$35	Zeiger plus Stringlänge
B69E	90 02	BCC \$B6A2	
B6A0	E6 36	INC \$36	
B6A2	60	RTS	
***** Stringverwaltung FRESTR			
B6A3	20 BF AD	JSR \$ADBF	prüft auf Stringvariable
B6A6	A5 64	LDA \$64	
B6A8	A4 65	LDY \$65	Zeiger auf Stringdescriptor
B6AA	85 22	STA \$22	
B6AC	84 23	STY \$23	
B6AE	20 DB B6	JSR \$B6DB	Descriptor vom Stringstack entfernen
B6B1	0B	PHP	
B6B2	A0 00	LDY \$00	
B6B4	B1 22	LDA (\$22),Y	
B6B6	4B	PHA	
B6B7	CB	INY	
B6B8	B1 22	LDA (\$22),Y	
B6BA	AA	TAX	
B6BB	CB	INY	
B6BC	B1 22	LDA (\$22),Y	
B6BE	AB	TAY	
B6BF	6B	PLA	
B6C0	2B	PLP	
B6C1	D0 13	BNE \$B6D6	String war nicht in Stringstack
B6C3	C4 34	CPY \$34	
B6C5	D0 0F	BNE \$B6D6	
B6C7	E4 33	CPX \$33	

B6C9	D0 0B	BNE \$B6D6	
B6CB	4B	PHA	
B6CC	1B	CLC	
B6CD	65 33	ADC \$33	
B6CF	85 33	STA \$33	\$33/\$34 zeigt jetzt auf Stringanfang
B6D1	90 02	BCC \$B6D5	
B6D3	E6 34	INC \$34	
B6D5	6B	PLA	
B6D6	86 22	STX \$22	
B6D8	84 23	STY \$23	
B6DA	60	RTS	
***** Stringzeiger aus Descriptorstack entfernen			
B6DB	C4 1B	CPY \$1B	
B6DD	D0 0C	BNE \$B6EB	
B6DF	C5 17	CHP \$17	1st Stringdescriptor in Stringstack ?
B6E1	D0 0B	BNE \$B6EB	
B6E3	85 16	STA \$16	
B6E5	E9 03	SBC #\$03	ja, Eintrag entfernen
B6E7	85 17	STA \$17	
B6E9	A0 00	LDY #\$00	
B6EB	60	RTS	
***** BASIC-Funktion CHR\$			
B6EC	20 A1 B7	JSR \$B7A1	holt Byte-Wert (0 bis 255)
B6EF	8A	TXA	Kode in Akku
B6F0	4B	PHA	
B6F1	A9 01	LDA #\$01	Länge des Strings gleich 1
B6F3	20 7D B4	JSR \$B47D	Platz für String reservieren
B6F6	6B	PLA	ASCII-Kode zurückholen
B6F7	A0 00	LDY #\$00	
B6F9	91 62	STA (\$62),Y	als Stringzeichen speichern
B6FB	6B	PLA	
B6FC	6B	PLA	
B6FD	4C CA B4	JMP \$B4CA	Descriptor in Stringstack bringen
***** BASIC-Funktion LEFT\$			
B700	20 61 B7	JSR \$B761	Stringparameter und Länge vom Stack holen
B703	D1 50	CHP (\$50),Y	Länge mit LEFT\$-Parameter vergleichen
B705	9B	TYA	
B706	90 04	BCC \$B70C	LEFT\$-Parameter kleiner als Stringlänge ?
B708	B1 50	LDA (\$50),Y	Stringlänge
B70A	AA	TAX	
B70B	9B	TYA	
B70C	4B	PHA	
B70D	8A	TXA	
B70E	4B	PHA	
B70F	20 7D B4	JSR \$B47D	Platz für neuen String reservieren
B712	A5 50	LDA \$50	
B714	A4 51	LDY \$51	Zeiger auf Descriptor
B716	20 AA B6	JSR \$B6AA	FRESTR
B719	6B	PLA	
B71A	A8	TAY	
B71B	6B	PLA	Länge des neuen Strings
B71C	1B	CLC	
B71D	65 22	ADC \$22	plus Adresse des alten Strings
B71F	85 22	STA \$22	
B721	90 02	BCC \$B725	
B723	E6 23	INC \$23	
B725	9B	TYA	

B726	20 8C B6	JSR \$B6BC	neuen String in Stringbereich übertragen
B729	4C CA B4	JMP \$B4CA	Descriptor in Stringstack bringen

\*\*\*\*\* BASIC-Funktion RIGHTS

B72C	20 61 B7	JSR \$B761	Stringparameter und Länge von Stack holen
B72F	18	CLC	
B730	F1 50	SBC (\$50),Y	von Stringlänge abziehen
B732	49 FF	EOR \$FF	Nummer des ersten Elements im alten String
B734	4C 06 B7	JMP \$B706	weiter wie LEFTs

\*\*\*\*\* BASIC-Funktion MID\$

B737	A9 FF	LDA \$FF	
B739	B5 65	STA \$65	
B73B	20 79 00	JSR \$0079	CHRGET letztes Zeichen holen
B73E	C9 29	CMP \$29	' ) ' Klammer zu
B740	F0 06	BEQ \$B748	
B742	20 FD AE	JSR \$AEFD	prüft auf Komma
B745	20 9E B7	JSR \$B79E	holt Byte-Wert, zweiten Parameter
B748	20 61 B7	JSR \$B761	Stringparameter und Startposition holen
B74B	F0 4B	BEQ \$B79B	1. Parameter null, 'illegal quantity'
B74D	CA	DEX	
B74E	BA	TXA	
B74F	48	PHA	Nummer des ersten Elements im alten String
B750	18	CLC	
B751	A2 00	LDX \$00	
B753	F1 50	SBC (\$50),Y	Länge des alten Strings
B755	B0 B6	BCS \$B70D	kleiner erster MID-Parameter
B757	49 FF	EOR \$FF	neue Stringlänge
B759	C5 65	CMP \$65	
B75B	90 B1	BCC \$B70E	
B75D	A5 65	LDA \$65	
B75F	B0 AD	BCS \$B70E	unbedingter Sprung

\*\*\*\*\* Stringparameter numer. Wert von Stack holen

B761	20 F7 AE	JSR \$AEF7	prüft auf Klammer zu
B764	68	PLA	
B765	A8	TAY	Aufrufadresse merken
B766	68	PLA	
B767	B5 55	STA \$55	
B769	68	PLA	
B76A	68	PLA	
B76B	68	PLA	1. Parameter
B76C	AA	TAX	
B76D	68	PLA	
B76E	B5 50	STA \$50	
B770	68	PLA	Adresse low und high des Stringdescriptors
B771	B5 51	STA \$51	
B773	A5 55	LDA \$55	
B775	48	PHA	
B776	98	TYA	Aufrufadresse wieder auf Stack
B777	48	PHA	
B778	A0 00	LDY \$00	
B77A	8A	TXA	Länge, zweiter Parameter
B77B	60	RTS	

\*\*\*\*\* BASIC-Funktion LEN

B77C	20 B2 B7	JSR \$B7B2	FRESTR, Stringlänge holen
B77F	4C A2 B3	JMP \$B3A2	Byte-Wert nach Fließkommaformat wandeln

\*\*\*\*\* Stringparameter holen

B782	20 A3 B6	JSR \$B6A3	FRESTR, String holen, Länge in A
B785	A2 00	LDX #00	
B787	86 0D	STX \$0D	Typflag auf numerisch setzen
B789	AB	TAY	Länge in Y
B78A	60	RTS	
***** BASIC-Funktion ASC			
B78B	20 82 B7	JSR \$B782	String holen, Zeiger in \$22/\$23, Länge in Y
B78E	F0 08	BEQ \$B798	Länge gleich null, 'illegal quantity'
B790	A0 00	LDY #00	
B792	B1 22	LDA (\$22),Y	erstes Zeichen holen
B794	AB	TAY	ASCII-Kode
B795	4C A2 B3	JMP \$B3A2	nach Fließkomma wandeln
B798	4C 48 B2	JMP \$B248	'illegal quantity'
***** holt Byte-Wert nach X			
B799	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
B79E	20 8A AD	JSR \$AD8A	FRMNUM numerischen Wert nach FAC holen
B7A1	20 88 B1	JSR \$B188	prüft auf Bereich und wandelt nach Integer
B7A4	A6 64	LDX #64	High-Byte
B7A6	D0 F0	BNE \$B798	ungleich null, dann 'illegal quantity'
B7A8	A6 65	LDX #65	
B7AA	4C 79 00	JMP \$0079	CHRGOT letztes Zeichen holen
***** BASIC-Funktion VAL			
B7AD	20 82 B7	JSR \$B782	Stringadresse und Länge holen
B7B0	D0 03	BNE \$B7B5	Stringlänge ungleich null ?
B7B2	4C F7 B8	JMP \$B8F7	null in FAC
B7B5	A6 7A	LDX #7A	
B7B7	A4 78	LDY #78	Programanzeiger retten
B7B9	86 71	STX #71	
B7BB	84 72	STY #72	
B7BD	A6 22	LDX #22	
B7BF	86 7A	STX #7A	Stringanfangsadresse in Stringzeiger bringen
B7C1	18	CLC	
B7C2	65 22	ADC #22	
B7C4	85 24	STA #24	
B7C6	A6 23	LDX #23	Stringendadresse + 1
B7C8	86 78	STX #78	
B7CA	90 01	BCC \$B7CD	
B7CC	E8	INX	
B7CD	86 25	STX #25	
B7CF	A0 00	LDY #00	
B7D1	B1 24	LDA (\$24),Y	erstes Byte nach String
B7D3	48	PHA	retten
B7D4	98	TYA	
B7D5	91 24	STA (\$24),Y	und durch null ersetzen
B7D7	20 79 00	JSR \$0079	CHRGOT letztes Zeichen holen
B7DA	20 F3 BC	JSR \$BCF3	String in Fließkommazahl umwandeln
B7DD	68	PLA	Zeichen nach String
B7DE	A0 00	LDY #00	
B7E0	91 24	STA (\$24),Y	wieder zurücksetzen
B7E2	A6 71	LDX #71	
B7E4	A4 72	LDY #72	
B7E6	86 7A	STX #7A	Programanzeiger wieder zurückholen
B7E8	84 78	STY #78	
B7EA	60	RTS	
***** GETADR und GETBYT holt 16-Bit und 8-Bit-Wert			
B7EB	20 BA AD	JSR \$ADBA	FRMNUM holt numerischen Wert

B7EE	20 F7 B7	JSR \$B7F7	FAC nach Adressformat wandlen \$14/\$15
B7F1	20 FD AE	JSR \$AEFD	CHKCOM prüft auf Komma
B7F4	4C 9E B7	JMP \$B79E	holt Byte-Wert nach X

*****			GETADR FAC in positive 16-Bit-Zahl wandeln
B7F7	A5 66	LDA \$66	Vorzeichen
B7F9	30 9D	BMI \$B798	negativ, dann 'illegal quantity'
B7FB	A5 61	LDA \$61	Exponent \$5536 vergleichen
B7FF	B0 97	BCS \$B798	größer, dann 'illegal quantity'
B801	20 98 BC	JSR \$B898	FAC in Adressformat wandeln
B804	A5 64	LDA \$64	
B806	A4 65	LDY \$65	Wert holen
B808	B4 14	STY \$14	und nach \$14/\$15
B80A	B5 15	STA \$15	
B80C	60	RTS	

*****			BASIC-Funktion PEEK
B80D	A5 15	LDA \$15	
B80F	4B	PHA	Adresse \$14/\$15 sichern
B810	A5 14	LDA \$14	
B812	4B	PHA	
B813	20 F7 B7	JSR \$B7F7	FAC nach Adressformat wandeln
B816	A0 00	LDY \$00	
B818	B1 14	LDA (\$14),Y	Peek-Wert holen
B81A	AB	TAY	nach Y
B81B	6B	PLA	
B81C	B5 14	STA \$14	
B81E	6B	PLA	Adresse zuruckholen
B81F	B5 15	STA \$15	
B821	4C A2 B3	JMP \$B3A2	Y nach Fließkommaformat

*****			BASIC-Befehl POKE
B824	20 EB B7	JSR \$B7EB	Poke-Adresse und Wert holen
B827	BA	TXA	Poke-Wert in Akku
B828	A0 00	LDY \$00	
B82A	91 14	STA (\$14),Y	und in Speicher schreiben
B82C	60	RTS	

*****			BASIC-Befehl WAIT
B82D	20 EB B7	JSR \$B7EB	Adresse und Wert holen
B830	B6 49	STX \$49	
B832	A2 00	LDX \$00	Default für dritten Parameter
B834	20 79 00	JSR \$0079	CHRGET letztes Zeichen
B837	F0 03	BEQ \$B83C	kein dritter Parameter ?
B839	20 F1 B7	JSR \$B7F1	prüft auf Komma und holt Parameter
B83C	B6 4A	STX \$4A	
B83E	A0 00	LDY \$00	
B840	B1 14	LDA (\$14),Y	Wait-Adresse
B842	45 4A	EDR \$4A	
B844	25 49	AND \$49	logisch verknüpfen
B846	F0 FB	BEQ \$B840	weiter warten
B848	60	RTS	

*****			Arithmetik-Routinen
*****			FAC = FAC + 0.5
B849	A9 11	LDA \$11	
B84B	A0 BF	LDY \$BF	Zeiger auf Konstante 0.5
B84D	4C 67 B8	JMP \$B867	FAC = FAC + Konstante (A/Y)



```

***** Minus FAC = Konstante (A/Y) - FAC
B850 20 BC BA JSR $BABC Konstante (A/Y) nach ARB

***** Minus FAC = ARG - FAC
B853 A5 66 LDA $66
B855 49 FF EOR $FF Vorzeichen umdrehen
B857 B5 66 STA $66
B859 45 6E EOR $6E
B85B B5 6F STA $6F
B85D A5 61 LDA $61
B85F 4C 6A BB JMP $BB6A FAC = FAC + ARG

*****
B862 20 99 B9 JSR $B999 Exponenten von FAC und ARG angleichen
B865 90 3C BCC $B8A3

***** Plus FAC = Konstante (A/Y) + FAC
B867 20 BC BA JSR $BABC Konstante (A/Y) nach ARG

***** Plus FAC = FAC + ARG
B86A D0 03 BNE $B86F FAC ungleich null ?
B86C 4C FC BB JMP $B8FC nein, dann FAC = ARG
B86F A6 70 LDX $70
B871 B6 56 STX $56
B873 A2 69 LDX $69
B875 A5 69 LDA $69
B877 A8 TAY
B878 F0 CE BEQ $B848
B87A 38 SEC
B87B E5 61 SBC $61
B87D F0 24 BEQ $B8A3
B87F 90 12 BCC $B893
B881 B4 61 STY $61
B883 A4 6E LDY $6E
B885 B4 66 STY $66
B887 49 FF EOR $FF
B889 69 00 ADC $00
B88B A0 00 LDY $00
B88D B4 56 STY $56
B88F A2 61 LDX $61
B891 D0 04 BNE $B897
B893 A0 00 LDY $00
B895 B4 70 STY $70
B897 C9 F9 CMP $F9
B899 30 C7 BMI $B862
B89B A8 TAY
B89C A5 70 LDA $70
B89E 56 01 LSR $01,X
B8A0 20 B0 B9 JSR $B9B0
B8A3 24 6F BIT $6F
B8A5 10 57 BPL $B8FE
B8A7 A0 61 LDY $61
B8A9 E0 69 CPX $69
B8AB F0 02 BEQ $B8AF
B8AD A0 69 LDY $69
B8AF 38 SEC
B8B0 49 FF EOR $FF
B8B2 65 56 ADC $56
B8B4 B5 70 STA $70
B8B6 B9 04 00 LDA $0004,Y

```

B8B9	F5 04	SBC #04,X	
B8BB	B5 65	STA #65	
B8BD	B9 03 00	LDA #0003,Y	
B8C0	F5 03	SBC #03,X	
B8C2	B5 64	STA #64	
B8C4	B9 02 00	LDA #0002,Y	
B8C7	F5 02	SBC #02,X	
B8C9	B5 63	STA #63	
B8CB	B9 01 00	LDA #0001,Y	
B8CE	F5 01	SBC #01,X	
B8D0	B5 62	STA #62	
B8D2	B0 03	BCS #B8D7	
B8D4	20 47 B9	JSR #B947	Mantisse von FAC invertieren
B8D7	A0 00	LDY #000	
B8D9	98	TYA	
B8DA	18	CLC	
B8DB	A6 62	LDX #62	
B8DD	D0 4A	BNE #B929	
B8DF	A6 63	LDX #63	
B8E1	B6 62	STX #62	
B8E3	A6 64	LDX #64	
B8E5	B6 63	STX #63	
B8E7	A6 65	LDX #65	
B8E9	B6 64	STX #64	
B8EB	A6 70	LDX #70	
B8ED	B6 65	STX #65	
B8EF	B4 70	STY #70	
B8F1	69 08	ADC #08	
B8F3	C9 20	CMR #20	
B8F5	D0 E4	BNE #B8DB	
B8F7	A9 00	LDA #000	
B8F9	B5 61	STA #61	
B8FB	B5 66	STA #66	
B8FD	60	RTS	
B8FE	65 56	ADC #56	
B900	B5 70	STA #70	
B902	A5 B5	LDA #65	
B904	65 6D	ADC #6D	
B906	B5 65	STA #65	
B908	A5 64	LDA #64	
B90A	65 6C	ADC #6C	
B90C	B5 64	STA #64	
B90E	A5 63	LDA #63	
B910	65 5B	ADC #6B	
B912	B5 63	STA #63	
B914	A5 62	LDA #62	
B916	65 6A	ADC #6A	
B918	B5 62	STA #62	
B91A	4C 36 B9	JMP #B936	
B91D	69 01	ADC #01	
B91F	06 70	ASL #70	
B921	26 65	ROL #65	
B923	26 64	ROL #64	
B925	26 63	ROL #63	
B927	26 62	ROL #62	
B929	10 F2	BPL #B91D	
B92B	38	SEC	
B92C	E5 61	SBC #61	
B92E	B0 C7	BCS #B8F7	

B930	49 FF	EOR #\$FF
B932	69 01	ADC #\$01
B934	85 61	STA \$61
B936	90 0E	BCC \$B946
B938	E6 61	INC \$61
B93A	F0 42	BEQ \$B97E
B93C	66 62	ROR \$62
B93E	66 63	ROR \$63
B940	66 64	ROR \$64
B942	66 65	ROR \$65
B944	66 70	ROR \$70
B946	60	RTS

+++++ Mantisse von FAC invertieren

B947	A5 66	LDA \$66
B949	49 FF	EOR #\$FF
B94B	85 66	STA \$66
B94D	A5 62	LDA \$62
B94F	49 FF	EOR #\$FF
B951	85 62	STA \$62
B953	A5 63	LDA \$63
B955	49 FF	EOR #\$FF
B957	85 63	STA \$63
B959	A5 64	LDA \$64
B95B	49 FF	EOR #\$FF
B95D	85 64	STA \$64
B95F	A5 65	LDA \$65
B961	49 FF	EOR #\$FF
B963	85 65	STA \$65
B965	A5 70	LDA \$70
B967	49 FF	EOR #\$FF
B969	85 70	STA \$70
B96B	E6 70	INC \$70
B96D	D0 0E	BNE \$B97D
B96F	E6 65	INC \$65
B971	D0 0A	BNE \$B97D
B973	E6 64	INC \$64
B975	D0 06	BNE \$B97D
B977	E6 63	INC \$63
B979	D0 02	BNE \$B97D
B97B	E6 62	INC \$62
B97D	60	RTS

überträge berücksichtigen

B97E	A2 0F	LDX #\$0F	Numer für 'overflow'
B980	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben

+++++ Rechtsverschieben eines Registers  
Offsetzeiger auf Register

B983	A2 25	LDX \$25
B985	B4 04	LDY \$04,X
B987	B4 70	STY \$70
B989	B4 03	LDY \$03,X
B98B	94 04	STY \$04,X
B98D	B4 02	LDY \$02,X
B98F	94 03	STY \$03,X
B991	B4 01	LDY \$01,X
B993	94 02	STY \$02,X
B995	A4 68	LDY \$68
B997	94 01	STY \$01,X
B999	69 08	ADC #\$08
B99B	30 EB	BMI \$B985

```

B99D F0 E6      BEQ $B9B5
B99F E9 08      SBC $08
B9A1 A8          TAY
B9A2 A5 70      LDA $70
B9A4 B0 14      BCS $B9BA
B9A6 16 01      ASL $01,X
B9A8 90 02      BCC $B9AC
B9AA F6 01      INC $01,X
B9AC 76 01      ROR $01,X
B9AE 76 01      ROR $01,X
B9B0 76 02      ROR $02,X
B9B2 76 03      ROR $03,X
B9B4 76 04      ROR $04,X
B9B6 6A          ROR A
B9B7 CB          INY
B9B8 D0 EC      BNE $B9A6
B9BA 1B          CLC
B9BB 60          RTS

```

\*\*\*\*\*

```

B9BC B1 00 00 00 00
B9C1 03
B9C2 7F 5E 56 CB 79
B9C7 B0 13 9B 0B 64
B9CC B0 76 38 93 16
B9D1 B2 38 AA 38 20
B9D5 B0 35 04 F3 34
B9DB B1 35 04 F3 34
B9E0 B0 B0 00 00 00
B9E5 B0 31 72 17 FB

```

Konstanten für LOG

```

1
3 = Polynomgrad, dann 4 Koeffizienten
.434255942
.576584541
.961800759
2.88539007
.707106781 = 1/SQR(2)
1.41421356 = SQR(2)
-.5
.693147181 = LOG(2)

```

\*\*\*\*\*

```

B9EA 20 2B BC    JSR $BC2B
B9ED F0 02      BED $B9F1
B9EF 10 03      BPL $B9F4
B9F1 4C 4B B2    JMP $B24B
B9F4 A5 61      LDA $61
B9F6 E9 7F      SBC $7F
B9F8 4B          PHA
B9F9 A9 B0      LDA $B0
B9FB B5 61      STA $61
B9FD A9 D6      LDA $D6
B9FF A0 B9      LDY $B9
BA01 20 67 BB    JSR $BB67
BA04 A9 DB      LDA $DB
BA06 A0 B9      LDY $B9
BA08 20 0F BB    JSR $BB0F
BA0B A9 BC      LDA $BC
BA0D A0 B9      LDY $B9
BA0F 20 50 BB    JSR $BB50
BA12 A9 C1      LDA $C1
BA14 A0 B9      LDY $B9
BA16 20 43 E0    JSR $E043
BA19 A9 E0      LDA $E0
BA1B A0 B9      LDY $B9
BA1D 20 67 BB    JSR $BB67
BA20 6B          PLA
BA21 20 7E BD    JSR $BD7E
BA24 A9 E5      LDA $E5
BA26 A0 B9      LDY $B9

```

BASIC-Funktion LOG

```

Vorzeichen holen
null, dann fertig
positiv, dann ok
'illegal quantity'
Exponent
normalisieren
und merken
Zahl in Bereich 0.5 bis 1
bringen

```

Zeiger auf Konstante 1/SQR(2)  
zu FAC addieren

Zeiger auf Konstante SQR(2)  
SQR(2) durch FAC dividieren

Zeiger auf Konstante 1  
1 minus FAC

Zeiger auf Polynomkoeffizienten  
Polynomberechnung

Zeiger auf Konstante -0.5  
zu FAC addieren  
Exponent zurückholen  
FAC = FAC + FAC

Zeiger auf Konstante LOG(2)

```

***** Multiplikation FAC = Konstante (A/Y) * FAC
BA2B 20 BC BA JSR $BABC Konstante nach ARG

***** Multiplikation FAC = ARG * FAC
BA2B D0 03 BNE $BA30 nicht null ?
BA2D 4C 8B BA JMP $BA8B RTS
BA30 20 B7 BA JSR $BAB7 Exponent berechnen
BA33 A9 00 LDA #00
BA35 B5 26 STA $26
BA37 B5 27 STA $27
BA39 B5 28 STA $28 Funktionsregister löschen
BA3B B5 29 STA $29
BA3D A5 70 LDA $70
BA3F 20 59 BA JSR $BA59 bitweise Multiplikation
BA42 A5 65 LDA $65
BA44 20 59 BA JSR $BA59 bitweise Multiplikation
BA47 A5 64 LDA $64
BA49 20 59 BA JSR $BA59 bitweise Multiplikation
BA4C A5 63 LDA $63
BA4E 20 59 BA JSR $BA59 bitweise Multiplikation
BA51 A5 62 LDA $62
BA53 20 5E BA JSR $BA5E bitweise Multiplikation
BA56 4C BF BB JMP $BBBF Register nach FAC, linksbündig machen
***** bitweise Multiplikation
BA59 D0 03 BNE $BA5E
BA5B 4C B3 B9 JMP $B9B3 rechtsverschieben des Registers
BA5E 4A LSR A
BA5F 09 80 ORA #80
BA61 A8 TAY
BA62 90 19 BCC $BA7D
BA64 18 CLC
BA65 A5 29 LDA $29
BA67 65 60 ADC $60
BA69 B5 29 STA $29
BA6B A5 28 LDA $28
BA6D 65 6C ADC $6C
BA6F B5 28 STA $28
BA71 A5 27 LDA $27
BA73 65 6B ADC $6B
BA75 B5 27 STA $27
BA77 A5 26 LDA $26
BA79 65 6A ADC $6A
BA7B B5 26 STA $26
BA7D 66 26 ROR $26
BA7F 66 27 ROR $27
BA81 66 28 ROR $28
BA83 66 29 ROR $29
BA85 66 70 ROR $70
BA87 9B TYA
BA8B 4A LSR A
BA89 D0 D6 BNE $BA61
BA8B 60 RTS

***** ARG = Konstante (A/Y)
BA8C B5 22 STA $22
BA8E B4 23 STY $23 Zeiger
BA90 A0 04 LDY #04
BA92 B1 22 LDA ($22),Y
BA94 B5 6D STA $6D
BA96 B8 DEY

```

BA97	B1 22	LDA (\$22),Y	
BA99	85 6C	STA \$6C	
BA9B	88	DEY	Konstante nach ARG
BA9C	B1 22	LDA (\$22),Y	
BA9E	85 6B	STA \$6B	
BAA0	88	DEY	
BAA1	B1 22	LDA (\$22),Y	
BAA3	85 6E	STA \$6E	
BAA5	45 66	EOR \$66	
BAA7	85 6F	STA \$6F	
BAA9	A5 6E	LDA \$6E	
BAA8	09 80	ORA #\$80	Vorzeichen
BAAD	85 6A	STA \$6A	
BAAF	88	DEY	
BAB0	B1 22	LDA (\$22),Y	
BAB2	85 69	STA \$69	Exponent
BAB4	A5 61	LDA \$61	
BAB6	60	RTS	
BAB7	A5 69	LDA \$69	
BAB9	F0 1F	BED \$BADA	
BABB	18	CLC	
BABC	65 61	ADC \$61	
BABE	90 04	BCC \$EAC4	
BAC0	30 1D	BMI \$BADF	
BAC2	18	CLC	
BAC3	2C	.BYTE \$2C	
BAC4	10 14	BPL \$BADA	
BAC6	69 80	ADC #\$80	
BAC8	85 61	STA \$61	
BACA	D0 03	BNE \$BACF	
BACC	4C FB 88	JMP \$BBFB	FAC = 0
BACF	A5 6F	LDA \$6F	
BAD1	85 66	STA \$66	
BAD3	60	RTS	
BAD4	A5 66	LDA \$66	
BAD6	49 FF	EOR \$FF	
BADB	30 05	BMI \$BADF	
BADA	68	PLA	
BADB	68	PLA	
BADC	4C F7 88	JMP \$BBF7	FAC = 0
BADF	4C 7E 89	JMP \$B97E	'overflow error'
***** FAC = FAC * 10			
BAE2	20 0C BC	JSR \$BC0C	FAC runden und nach ARG
BAE5	AA	TAX	
BAE6	F0 10	BEQ \$BAFB	FAC gleich null, dann fertig
BAE8	18	CLC	
BAE9	69 02	ADC #\$02	Exponent + 2 entspricht mal 4
BAEB	B0 F2	BCS \$BADF	Übertrag ?
BAED	A2 00	LDX #\$00	
BAEF	B6 6F	STX \$6F	
BAF1	20 77 88	JSR \$BB77	FAC = FAC + ARG entspricht mal 5
BAF4	E6 61	INC \$61	Exponent erhöhen entspricht mal 2
BAF6	F0 E7	BEQ \$BADF	übertrag, dann 'overflow'
BAF8	60	RTS	
*****			
BAF9	B4 20 00 00 00		Fließkommakonstante 10
***** FAC = FAC / 10			

BABE	20 0C BC	JSR \$B00C	FAC runden und nach ARG
BB01	A9 F9	LDA #\$F9	
BB03	A0 BA	LDY #\$BA	Zeiger auf Konstante 10
BB05	A2 00	LDX #\$00	
BB07	B6 6F	STX \$6F	
BB09	20 A2 BB	JSR \$BBA2	Konstante 10 nach FAC
BB0C	4C 12 BB	JMP \$BB12	FAC = ARG / FAC
*****			FAC = Konstante (A/Y) / FAC
BB0F	20 BC BA	JSR \$BABC	Konstante (A/Y) nach ARG
*****			FAC = ARG / FAC
BB12	F0 76	BEQ \$BB8A	FAC gleich null, 'division by zero'
BB14	20 1B BC	JSR \$B0C1B	FAC runden
BB17	A9 00	LDA \$00	
BB19	3B	SEC	
BB1A	E5 61	SBC \$61	
BB1C	B5 61	STA \$61	
BB1E	20 B7 BA	JSR \$BAB7	Exponent des Ergebnisses bestimmen
BB21	E6 61	INC \$61	
BB23	F0 BA	BEQ \$BADF	Exponentenuberlauf, 'overflow'
BB25	A2 FC	LDX \$FC	Zeiger auf Funktionsregister
BB27	A9 01	LDA \$01	
BB29	A4 6A	LDY \$6A	
BB2B	C4 62	CPY \$62	
BB2D	D0 10	BNE \$BB3F	
BB2F	A4 6B	LDY \$6B	
BB31	C4 63	CPY \$63	ARG mit FAC byteweise vergleichen
BB33	D0 0A	BNE \$BB3F	
BB35	A4 6C	LDY \$6C	
BB37	C4 64	CPY \$64	
BB39	D0 04	BNE \$BB3F	
BB3B	A4 6D	LDY \$6D	
BB3D	C4 65	CPY \$65	
BB3F	0B	PHP	Status retten
BB40	2A	ROL A	
BB41	90 09	BCC \$BB4C	
BB43	E8	INX	
BB44	95 29	STA \$29,X	
BB46	F0 32	BEQ \$BB7A	
BB48	10 34	BPL \$BB7E	
BB4A	A9 01	LDA \$01	
BB4C	2B	PLP	
BB4D	B0 0E	BCS \$BB5D	
BB4F	06 6D	ASL \$6D	
BB51	26 6C	ROL \$6C	
BB53	26 6B	ROL \$6B	
BB55	26 6A	ROL \$6A	
BB57	B0 E6	BCS \$BB3F	
BB59	30 CE	BNL \$BB29	
BB5B	10 E2	BPL \$BB3F	
BB5D	AB	TAY	
BB5E	A5 6D	LDA \$6D	
BB60	E5 65	SBC \$65	
BB62	B5 6D	STA \$6D	
BB64	A5 6C	LDA \$6C	
BB66	E5 64	SBC \$64	
BB68	B5 6C	STA \$6C	
BB6A	A5 6B	LDA \$6B	
BB6C	E5 63	SBC \$63	

BB6E	B5 6B	STA \$6B	
BB70	A5 6A	LDA \$6A	
BB72	E5 62	SBC \$62	
BB74	B5 6A	STA \$6A	
BB76	9B	TYA	
BB77	4C 4F BB	JMP \$BB4F	
BB7A	A9 40	LDA \$40	
BB7C	D0 CE	BNE \$BB4C	
BB7E	0A	ASL A	
BB7F	0A	ASL A	
BB80	0A	ASL A	Akku * 64
BB81	0A	ASL A	
BB82	0A	ASL A	
BB83	0A	ASL A	
BB84	B5 70	STA \$70	
BB86	2B	PLP	
BB87	4C BF BB	JMP \$BBBF	Hilfsregister nach FAC
*****			
BB8A	A2 14	LDX \$14	Numer für 'division by zero'
BB8C	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
***** Hilfsregister (\$26 - \$29) nach FAC übertragen			
BB8F	A5 26	LDA \$26	
BB91	B5 62	STA \$62	
BB93	A5 27	LDA \$27	
BB95	B5 63	STA \$63	
BB97	A5 28	LDA \$28	
BB99	B5 64	STA \$64	
BB9B	A5 29	LDA \$29	
BB9D	B5 65	STA \$65	
BB9F	4C D7 BB	JMP \$BBD7	FAC linksbündig machen
***** Konstante (A/Y) nach FAC übertragen			
BBA2	B5 22	STA \$22	
BBA4	B4 23	STY \$23	Zeiger setzen
BBA6	A0 04	LDY \$04	
BBA8	B1 22	LDA (\$22),Y	
BBA A	B5 65	STA \$65	
BBA C	8B	DEY	
BBA D	B1 22	LDA (\$22),Y	
BBA F	B5 64	STA \$64	
BBB1	8B	DEY	Mantisse
BBB2	B1 22	LDA (\$22),Y	
BBB4	B5 63	STA \$63	
BBB6	8B	DEY	
BBB7	B1 22	LDA (\$22),Y	
BBB9	B5 66	STA \$66	
BBB B	09 80	ORA \$80	Vorzeichen Mantisse
BBB D	B5 62	STA \$62	
BBB F	8B	DEY	
BBC0	B1 22	LDA (\$22),Y	
BBC2	B5 61	STA \$61	Exponent
BBC4	B4 70	STY \$70	
BBC6	60	RTS	
***** FAC nach Akku#4 übertragen			
BBC7	A2 5C	LDX \$5C	Adresse low Akku#4
BBC9	2C	.BYTE \$2C	



```

***** FAC nach Akku#3 übertragen
BBCA A2 57 LDX #57 Adresse low Akku#3
BBCC A0 00 LDY #00 Adresse high
BBCE F0 04 BEQ $BBD4 unbedingter Sprung

***** FAC nach Variable übertragen
BBD0 A6 49 LDX $49
BBD2 A4 4A LDY $4A Variablenadresse
BBD4 20 1B BC JSR $BC1B FAC runden
BBD7 86 22 STX $22
BBD9 84 23 STY $23 Zeiger auf Zieladresse
BBD8 A0 04 LDY #04
BBD0 A5 65 LDA $65
BBD0F 91 22 STA ($22),Y
BBE1 88 DEY
BBE2 A5 64 LDA $64
BBE4 91 22 STA ($22),Y
BBE6 88 DEY
BBE7 A5 63 LDA $63
BBE9 91 22 STA ($22),Y
BBEB 88 DEY
BBEC A5 66 LDA $66
BBEE 09 7F ORA #$7F
BBF0 25 62 AND $62 Vorzeichen auf Speicherformat bringen
BBF2 91 22 STA ($22),Y
BBF4 88 DEY
BBF5 A5 61 LDA $61
BBF7 91 22 STA ($22),Y
BBF9 84 70 STY $70
BBFB 60 RTS

***** ARG nach FAC übertragen
BBFC A5 6E LDA $6E
BBFE 85 66 STA $66
BC00 A2 05 LDX #05 5 Bytes
BC02 85 6E LDA $6E,X
BC04 95 60 STA $60,X
BC06 CA DEX
BC07 D0 F9 BNE $BC02
BC09 86 70 STX $70
BC0B 60 RTS

***** FAC nach ARG übertragen
BC0C 20 1B BC JSR $BC1B FAC runden
BC0F A2 06 LDX #06
BC11 85 60 LDA $60,X
BC13 95 68 STA $68,X
BC15 CA DEX
BC16 D0 F9 BNE $BC11
BC18 86 70 STX $70
BC1A 60 RTS

***** FAC runden
BC1B A5 61 LDA $61 Exponent null, dann fertig
BC1D F0 F8 BEQ $BC1A
BC1F 06 70 ASL $70 Rundungsstelle größer $7F ?
BC21 90 F7 BCC $BC1A nein, dann fertig
BC23 20 6F B9 JSR $B96F Mantisse um eins erhöhen
BC26 D0 F2 BNE $BC1A jetzt null ?
BC28 4C 3B B9 JMP $B93B nach rechts verschieben, Exponent erhöhen

```

```

***** Vorzeichen von FAC holen
BC2B A5 61 LDA $61 null ?
BC2D F0 0F BEQ $BC3B
BC2F A5 66 LDA $66
BC31 2A ROL A
BC32 A9 FF LDA $FF negativ
BC34 B0 02 BCS $BC3B
BC36 A9 01 LDA $01 positiv
BC3B 60 RTS

***** BASIC-Funktion SGN
BC39 20 2B BC JSR $BC2B Vorzeichen holen
BC3C B5 62 STA $62
BC3E A9 00 LDA $00
BC40 B5 63 STA $63
BC42 A2 68 LDX $68
BC44 A5 62 LDA $62
BC46 A9 FF EOR $FF
BC48 2A ROL A
BC49 A9 00 LDA $00
BC4B B5 65 STA $65
BC4D B5 64 STA $64
BC4F B6 61 STX $61
BC51 B5 70 STA $70
BC53 B5 66 STA $66
BC55 4C D2 BB JMP $BBD2

***** BASIC-Funktion ABS
BC5B 46 66 LSR $66 Vorzeichenbit löschen
BC5A 60 RTS

***** Vergleich Konstante (A/Y) mit FAC
BC5B B5 24 STA $24
BC5D B4 25 STY $25 Zeiger auf Konstante
BC5F A0 00 LDY $00
BC61 B1 24 LDA ($24),Y Exponent
BC63 C8 INY
BC64 A4 TAX
BC65 F0 C4 BEQ $BC2B null, dann Vorzeichen von FAC holen
BC67 B1 24 LDA ($24),Y
BC69 45 66 EOR $66
BC6B 30 C2 BMI $BC2F verschiedene Vorzeichen
BC6D E4 61 CPX $61
BC6F D0 21 BNE $BC92
BC71 B1 24 LDA ($24),Y 1. Byte vergleichen
BC73 09 B0 ORA $B0
BC75 C5 62 CMP $62
BC77 D0 19 BNE $BC92
BC79 C8 INY
BC7A B1 24 LDA ($24),Y 2. Byte vergleichen
BC7C C5 63 CMP $63
BC7E D0 12 BNE $BC92
BC80 C8 INY
BC81 B1 24 LDA ($24),Y 3. Byte vergleichen
BC83 C5 64 CMP $64
BC85 D0 0B BNE $BC92
BC87 C8 INY
BC89 A9 7F LDA $7F
BC8A C5 70 CMP $70
BC8C B1 24 LDA ($24),Y 4. Byte vergleichen

```

BC8E	E5 65	SBC \$65	
BC90	F0 28	BEQ \$BCBA	
BC92	A5 66	LDA \$66	
BC94	90 02	BCC \$BC98	
BC96	49 FF	EOR \$FF	Ergebnis kleiner, dann invertieren
BC98	4C 31 BC	JMP \$BC31	Flag für Ergebnis setzen

\*\*\*\*\* Umwandlung Fließkomma nach Integer  
 Exponent  
 null ?

BC9B	A5 61	LDA \$61	
BC9D	F0 4A	BEQ \$BCE9	
BC9F	3B	SEC	
BCA0	E9 A0	SBC \$A0	
BCA2	24 66	BIT \$66	
BCA4	10 09	BPL \$BCAF	
BCA6	AA	TAX	
BCA7	49 FF	LDA \$FF	
BCA9	B5 68	STA \$68	
BCAB	20 4D B9	JSR \$B94D	Mantisse von FAC invertieren
BCAE	8A	TXA	
BCAF	A2 61	LDX \$61	
BCB1	C9 F9	CMF \$F9	
BCB3	10 06	BPL \$BCBB	
BCB5	20 99 B9	JSR \$B999	FAC rechtsverschieben
BCB8	B4 68	STY \$68	
BCBA	60	RTS	

BCBB	A8	TAY	
BCBC	A5 66	LDA \$66	
BCBE	29 80	AND \$80	
BCC0	46 62	LSR \$62	
BCC2	05 62	ORA \$62	
BCC4	B5 62	STA \$62	
BCC6	20 80 B9	JSR \$B9B0	FAC bitweise nach rechts verschieben
BCC9	B4 68	STY \$68	
BCCB	60	RTS	

\*\*\*\*\* BASIC-Funktion INT  
 Exponent  
 ganze Zahl ?  
 ja, dann fertig  
 FAC nach Integer wandeln

BCCC	A5 61	LDA \$61	
BCCE	C9 A0	CMF \$A0	
BCD0	B0 20	BCS \$BCF2	
BCD2	20 9B BC	JSR \$BC9B	
BCD5	B4 70	STY \$70	
BCD7	A5 66	LDA \$66	
BCD9	B4 66	STY \$66	
BCDB	49 80	EOR \$80	
BCDD	2A	ROL A	
BCDE	A9 A0	LDA \$A0	
BCE0	B5 61	STA \$61	
BCE2	A5 65	LDA \$65	
BCE4	B5 07	STA \$07	
BCE6	4C D2 B8	JMP \$BBD2	FAC linksbündig machen

BCE9	B5 62	STA \$62	Mantisse mit nullen füllen
BCEB	B5 63	STA \$63	
BCED	B5 64	STA \$64	
BCEF	B5 65	STA \$65	
BCF1	A8	TAY	
BCF2	60	RTS	

\*\*\*\*\* Umwandlung ASCII nach Fließkommaformat

BCF3	A0 00	LDY #00	
BCF5	A2 0A	LDX #0A	Bereich \$5D bis \$66 löschen
BCF7	94 5D	STY \$5D,X	
BCF9	CA	DEX	
BCFA	10 FB	BPL \$BCF7	
BCFC	90 0F	BCC \$BD0D	
BCFE	C9 2D	CMP #2D	'-'
BD00	D0 04	BNE \$BD06	
BD02	B6 67	STX \$67	Flag für negativ
BD04	F0 04	BEQ \$BD0A	
BD06	C9 2B	CMP #2B	'+'
BD08	D0 05	BNE \$BD0F	
BD0A	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
BD0D	90 5B	BCC \$BD6A	
BD0F	C9 2E	CMP #2E	'.'
BD11	F0 2E	BEQ \$BD41	
BD13	C9 45	CMP #45	'E'
BD15	D0 30	BNE \$BD47	
BD17	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
BD1A	90 17	BCC \$BD33	
BD1C	C9 AB	CMP #AB	'-' BASIC-Kode
BD1E	F0 0E	BEQ \$BD2E	
BD20	C9 2D	CMP #2D	'-'
BD22	F0 0A	BEQ \$BD2E	
BD24	C9 AA	CMP #AA	'=' BASIC-Kode
BD26	F0 0B	BEQ \$BD30	
BD28	C9 2B	CMP #2B	'+'
BD2A	F0 04	BEQ \$BD30	
BD2C	D0 07	BNE \$BD35	
BD2E	66 60	ROR \$60	Bit 7 setzen
BD30	20 73 00	JSR \$0073	CHRGET nächstes Zeichen holen
BD33	90 5C	BCC \$BD91	
BD35	24 60	BIT \$60	Bit 7 gesetzt ?
BD37	10 0E	BPL \$BD47	nein
BD39	A9 00	LDA #00	
BD3B	3B	SEC	
BD3C	E5 5E	SBC \$5E	
BD3E	4C 49 BD	JMP \$BD49	
BD41	66 5F	ROR \$5F	Aufruf durch Dezimalpunkt
BD43	24 5F	BIT \$5F	
BD45	50 C3	BVC \$BD0A	
BD47	A5 5E	LDA \$5E	
BD49	3B	SEC	
BD4A	E5 5D	SBC \$5D	
BD4C	85 5E	STA \$5E	
BD4E	F0 12	BEQ \$BD62	
BD50	10 09	BPL \$BD5B	
BD52	20 FE BA	JSR \$BAFE	FAC = FAC / 10
BD55	E6 5E	INC \$5E	
BD57	D0 F9	BNE \$BD52	
BD59	F0 07	BEQ \$BD62	
BD5B	20 E2 BA	JSR \$BAE2	FAC = FAC * 10
BD5E	C6 5E	DEC \$5E	
BD60	D0 F9	BNE \$BD5B	
BD62	A5 67	LDA \$67	
BD64	30 01	BMI \$BD67	
BD66	60	RTS	
BD67	4C B4 BF	JMP \$BFB4	Vorzeichenwechsel FAC = - FAC
BD6A	4B	PHA	

BD6B	24 5F	BIT \$5F	
BD6D	10 02	BPL \$BD71	
BD6F	E6 5D	INC \$5D	
BD71	20 E2 BA	JSR \$BAE2	FAC = FAC * 10
BD74	6B	PLA	
BD75	3B	SEC	
BD76	E9 30	SBC #\$30	'0' abziehen gibt hex
BD78	20 7E BD	JSR \$BD7E	addiert nächste Stelle zu FAC
BD7B	4C 0A BD	JMP \$BD0A	nächstes Zeichen
BD7E	4B	PHA	
BD7F	20 0C BC	JSR \$BC0C	FAC nach AR6
BD82	6B	PLA	
BD83	20 3C BC	JSR \$BC3C	
BD86	A5 6E	LDA \$6E	
BD88	45 66	EOR \$66	
BD8A	85 6F	STA \$6F	
BD8C	A6 61	LDX \$61	
BD8E	4C 6A B8	JMP \$BB6A	FAC = FAC + AR6
BD91	A5 5E	LDA \$5E	Aufruf durch 'E'
BD93	C9 0A	CMR \$0A	
BD95	90 09	BCC \$BDA0	
BD97	A9 64	LDA #\$64	
BD99	24 60	BIT \$60	
BD9B	30 11	BMI \$BDAE	
BD9D	4C 7E B9	JMP \$B97E	'overflow error'
BDA0	0A	ASL A	
BDA1	0A	ASL A	
BDA2	1B	CLC	
BDA3	65 5E	ADC \$5E	
BDA5	0A	ASL A	
BDA6	1B	CLC	
BDA7	A0 00	LDY #\$00	
BDA9	71 7A	ADC (\$7A),Y	
BDA8	3B	SEC	
BDAC	E9 30	SBC #\$30	'0'
BDAE	85 5E	STA \$5E	
BDB0	4C 30 BD	JMP \$BD30	nächstes Zeichen holen
*****		Konstanten für Fließkomma nach ASCII	
BDB3	9B 3E BC 1F FD	99999999.9	
BDBB	9E 6E 6B 27 FD	999999999	
BDBD	9E 6E 6B 2B 00	1E9	
*****		Ausgabe der Zeilennummer bei Fehlermeldung	
BDC2	A9 71	LDA #\$71	
BDC4	A0 A3	LDY \$A3	Zeiger auf 'in'
BDC6	20 DA BD	JSR \$BDDA	String ausgeben
BDC9	A5 3A	LDA \$3A	
BDCB	A6 39	LDX \$39	laufende Zeilennummer holen
*****		positive Integerzahl in A/X ausgeben	
BDCD	B5 62	STA \$62	
BDCF	86 63	STX \$63	für Umwandlung in FAC schreiben
BDD1	A2 90	LDX \$90	
BDD3	3B	SEC	
BDD4	20 49 BC	JSR \$BC49	Integer nach Fließkomma wandeln
BDD7	20 DF BD	JSR \$BDDF	FAC nach ASCII wandeln
BDDA	4C 1E AB	JMP \$AB1E	String ausgeben

```

***** FAC nach ASCII-Format wandeln und nach $100
BDD0 A0 01 LDY #01
BDDF A9 20 LDA #20      ' ' Leerzeichen für positive Zahl
BDE1 24 66 BIT #66      Vorzeichen
BDE3 10 02 BPL #BDE7    positiv ?
BDE5 A9 20 LDA #20      '-' Minuszeichen für negative Zahl
BDE7 99 FF 00 STA #00FF,Y in Pufferbereich schreiben
BDEA 85 66 STA #66
BDEC 84 71 STY #71
BDEE CB INY
BDEF A9 30 LDA #30      '0'
BDF1 A6 61 LDX #61      Exponent
BDF3 D0 03 BNE #BDF8    Zahl nicht null ?
BDF5 4C 04 BF JMP #BFO4  ja, fertig
BDF8 A9 00 LDA #00
BDFA E0 80 CPX #80      FAC mit 1 vergleichen
BDFC F0 02 BEQ #BE00
BDFE 80 09 BCS #BE09    FAC größer 1
BE00 A9 8D LDA #8D
BE02 A0 8D LDY #8D      Zeiger auf Konstante 1E9
BE04 20 28 BA JSR #BA28 Konstante (Zeiger A/Y) * FAC
BE07 A9 F7 LDA #F7
BE09 85 5D STA #5D
BE0B A9 8B LDA #8B
BE0D A0 8D LDY #8D      Zeiger auf Konstante 99999999
BE0F 20 5B BC JSR #BC5B Vergleich Konstante (Zeiger A/Y) mit FAC
BE12 F0 1E BEQ #BE32    gleich
BE14 10 12 BPL #BE2B
BE16 A9 B3 LDA #B3
BE18 A0 8D LDY #8D      Zeiger auf Konstante 99999999,9
BE1A 20 5B BC JSR #BC5B Vergleich Konstante (Zeiger A/Y) mit FAC
BE1D F0 02 BEQ #BE21
BE1F 10 0E BPL #BE2F
BE21 20 E2 BA JSR #BAE2  FAC = FAC * 10
BE24 C6 5D DEC #5D
BE26 D0 EE BNE #BE16
BE28 20 FE BA JSR #BAFE  FAC = FAC / 10
BE2B E6 5D INC #5D
BE2D D0 DC BNE #BE0B
BE2F 20 49 BB JSR #BB49  FAC = FAC + .5 , runden
BE32 20 9B BC JSR #BC9B  FAC nach Integer
BE35 A2 01 LDX #01
BE37 A5 5D LDA #5D
BE39 18 CLC
BE3A 69 0A ADC #0A
BE3C 30 09 BMI #BE47    Betrag kleiner 0.1 ?
BE3E C9 0B CMP #0B
BE40 B0 06 BCS #BE48    Betrag größer 1E9 ?
BE42 69 FF ADC #FF
BE44 AA TAX
BE45 A9 02 LDA #02
BE47 38 SEC
BE48 E9 02 SBC #02
BE4A 85 5E STA #5E
BE4C 86 5D STX #5D
BE4E 8A TXA
BE4F F0 02 BEQ #BE53
BE51 10 13 BPL #BE66
BE53 A4 71 LDY #71
BE55 A9 2E LDA #2E

```

BE57	CB	INY	
BE58	99 FF 00	STA \$00FF,Y	
BE5B	BA	TXA	
BE5C	F0 06	BEO \$BE64	
BE5E	A9 30	LDA #30	10
BE60	CB	INY	
BE61	99 FF 00	STA \$00FF,Y	
BE64	84 71	STY \$71	
BE66	A0 00	LDY #00	Berechnung der einzelnen Ziffern
BE68	A2 80	LDX #80	
BE6A	A5 65	LDA \$65	
BE6C	18	CLC	
BE6D	79 19 BF	ADC \$BF19,Y	
BE70	B5 65	STA \$65	
BE72	A5 64	LDA \$64	
BE74	79 18 BF	ADC \$BF18,Y	
BE77	B5 64	STA \$64	
BE79	A5 63	LDA \$63	
BE7B	79 17 BF	ADC \$BF17,Y	
BE7E	B5 63	STA \$63	
BE80	A5 62	LDA \$62	
BE82	79 16 BF	ADC \$BF16,Y	
BE85	B5 62	STA \$62	
BE87	E8	INX	
BE88	B0 04	BCC #BE8E	
BE8A	10 DE	BPL \$BE6A	
BE8C	30 02	BMI \$BE90	
BE8E	30 DA	BMI \$BE6A	
BE90	BA	TXA	
BE91	90 04	BCC \$BE97	
BE93	49 FF	EOR #FF	
BE95	69 0A	ADC #0A	10
BE97	69 2F	ADC #2F	10 - 1
BE99	CB	INY	
BE9A	CB	INY	
BE9B	CB	INY	
BE9C	CB	INY	
BE9D	B4 47	STY \$47	
BE9F	A4 71	LDY \$71	
BEA1	CB	INY	
BEA2	AA	TAX	
BEA3	29 7F	AND #7F	
BEA5	99 FF 00	STA \$00FF,Y	
BEA8	C6 5D	DEC \$5D	
BEAA	D0 06	BNE \$BEB2	
BEAC	A9 2E	LDA #2E	
BEAE	CB	INY	
BEAF	99 FF 00	STA \$00FF,Y	
BEB2	B4 71	STY \$71	
BEB4	A4 47	LDY \$47	
BEB6	BA	TXA	
BEB7	49 FF	EOR #FF	
BEB9	29 80	AND #80	
BEBB	AA	TAX	
BEBC	C0 24	CPY #24	Tabellenende bei FAC-Umwandlung
BEBE	F0 04	BEC \$BEC4	
BEC0	C0 3C	CPY #3C	Tabellenende bei TIS-Berechnung
BEC2	D0 A6	BNE \$BE6A	
BEC4	A4 71	LDY \$71	
BEC6	B9 FF 00	LDA \$00FF,Y	

BEC9	88	DEY	
BEC4	C9 30	CMF #30	0
BEC0	F0 F8	BEG #BEC6	
BECE	C9 2E	CMF #32E	
BED0	F0 01	BEG #BED3	
BED2	08	INY	
BED3	A9 2B	LDA #32B	*
BED5	A6 5E	LDX #5E	
BED7	F0 2E	BEG #BF07	
BED9	10 0B	BPL #BEE3	
BEDB	A9 00	LDA #00	
BEDD	3B	SEC	
BEDF	E5 5E	SBC #5E	
BEE0	AA	TAX	
BEE1	A9 2D	LDA #32D	-
BEE3	99 01 01	STA #0101.Y	
BEE6	A9 45	LDA #45	E
BEEB	99 00 01	STA #0100.Y	
BEEB	8A	TXA	
BEEC	A2 2F	LDX #32F	0 - 1
BEEE	3B	SEC	
BEEF	EB	INY	
BEF0	E9 04	SEC #30A	10
BEF2	B0 FB	BCS #BEEF	
BEF4	69 3A	ADC #3A	9 - 1
BEF6	99 03 01	STA #0103.Y	
BEF9	8A	TXA	
BEFA	99 02 01	STA #0102.Y	
BEFD	A9 00	LDA #00	Puffer mit 30 abschließen
BEFF	99 04 01	STA #0104.Y	
BF02	F0 0B	BEG #BF0C	
BF04	99 FF 00	STA #00FF.Y	
BF07	A9 00	LDA #00	Puffer mit 30 abschließen
BF09	99 00 01	STA #0100.Y	
BF0C	A9 00	LDA #00	
BF0E	A0 01	LDY #01	Zeiger auf Puffer 3100
BF10	60	RTS	
*****			
BF11	80 00 00 00 00	Konstante 0.5 für SDR-Funktion	
*****			
		Konstanten für Gleitkomma nach ASCII	
		32-Bit Binärzahlen mit Vorzeichen	
BF16	FA 0A 1F 00	-100 000 000	
BF1A	00 9B 96 B0	10 000 000	
BF1E	FF F0 BD C0	-1 000 000	
BF22	00 01 B6 A0	100 000	
BF26	FF FF DB F0	-10 000	
BF2A	00 00 03 E8	1 000	
BF2E	FF FF FF 9C	- 100	
BF32	00 00 00 0A	10	
BF36	FF FF FF FF	-1	
*****			
		Konstanten für Umwandlung TI nach TIS	
BF3A	FF DF 0A B0	-2 160 000	
BF3E	00 03 4B C0	216 000	
BF42	FF FF 73 60	-36 000	
BF46	00 00 0E 10	3 600	
BF4A	FF FF FD AB	- 600	
BF4E	00 00 00 3C	60	



BF52 EC

BF53 AA ...

BF70 ... AA

\*\*\*\*\*

BF71 20 0C BC JSR \$B0C0C

BF74 A9 11 LDA #11

BF76 A0 BF LDY #BF

BASIC-Funktion SOR

FAC runden und nach ARG

Zeiger auf Konstante 0.5

\*\*\*\*\*

BF76 20 A2 BB JSR \$BBA2

Potenzierung FAC = ARG hoch Konstante (A/Y)

Konstante nach FAC

\*\*\*\*\*

BF7B F0 70 BEQ \$BFED

BF7D A5 69 LDA \$69

BF7F D0 03 BNE \$BF84

BF81 4C F9 BB JMP \$BF99

BF84 A2 4E LDA #4E

BF86 A0 00 LDY #00

BF88 20 D4 BB JSR \$BBD4

BF8B A5 6E LDA \$6E

BF8D 10 0F BPL \$BF9E

BF8F 20 CC BC JSR \$BCCC

BF92 A9 4E LDA #4E

BF94 A0 00 LDY #00

BF96 20 5B BC JSR \$B05B

BF99 D0 03 BNE \$BF9E

BF9B 98 TYA

BF9C A4 C7 LDY \$07

BF9E 20 FE BB JSR \$B0FE

BFA1 98 TYA

BFA2 4B PHA

BFA3 20 EA B9 JSR \$B9EA

BFA6 A9 4E LDA #4E

BFA8 A0 00 LDY #00

BFAA 20 28 BA JSR \$BA28

BFAD 20 ED BF JSR \$BFED

BFB0 6B PLA

BFB1 4A LSR A

BFB2 90 0A BCC \$BFB2

Potenzierung FAC = ARG hoch FAC

Exponent ARG = Basis

nicht null ?

fertig

Zeiger auf Hilfsakku

FAC nach Hilfsakku

Exponent FAC = Potenzexponent

Kleiner eins ?

INT-Funktion

Zeiger auf Hilfsakku

mit FAC vergleichen

ARG nach FAC

LOG-Funktion

Zeiger auf Hilfsakku

mit FAC multiplizieren

EXP-Funktion

\*\*\*\*\*

BFB4 A5 61 LDA \$61

BFB6 F0 06 BEQ \$BFB2

BFB8 A5 66 LDA \$66

BFBA 49 FF EOR #\$FF

BFBC B5 66 STA \$66

BFBE 60 RTS

Vorzeichenwechsel

Exponent

Zahl gleich null, dann fertig

Vorzeichen invertieren

\*\*\*\*\*

BFBF B1 38 AA 3B 29

BFC4 07

BFC5 71 34 5B 3E 56

BFCA 74 16 7E B3 1B

BFCF 77 2F EE E3 83

BFD4 7A 1D B4 1C 2A

BFD9 7C 63 59 5B 0A

BFDE 7E 75 FD E7 C6

BFE3 B0 31 72 1B 10

Konstanten für EXP

1.44269504 = 1/LOG(2)

7 = Polynomgrad, 8 Koeffizienten

2.14987637E-5

1.4352314E-4

1.34226348E-3

9.614011701E-3

.0555051269

.240226385

.693147186

\*\*\*\*\*

BASIC-Funtion EXP

```

BFED A9 BF LDA #BF
9FEF A0 BF LDY #BF
BFF1 20 28 BA JSR $BA28
BFF4 A5 70 LDA $70
BFF6 69 50 ADC #50
BFFB 90 03 BCC $BFFD
BFFA 20 23 BC JSR $BC23
BFFD 4C 00 E0 JMP $E000
E000 85 56 STA $56
E002 20 0F BC JSR $BC0F
E005 A5 61 LDA $61
E007 C9 88 CMP #88
E009 90 03 BCC $E00E
E00B 20 D4 BA JSR $BAD4
E00E 20 CC BC JSR $BCCC
E011 A5 07 LDA $07
E013 18 CLC
E014 69 B1 ADC #B1
E016 F0 F3 BEQ $E00B
E01B 38 SEC
E019 E9 01 SBC #01
E01B 48 PHA
E01C A2 05 LDX #05
E01E B5 69 LDA $69,X
E020 B4 61 LDY $61,X
E022 95 61 STA $61,X
E024 94 69 STY $69,X
E026 CA DEX
E027 10 F5 BPL $E01E
E029 A5 56 LDA $56
E02B 85 70 STA $70
E02D 20 53 B8 JSR $B853
E030 20 B4 BF JSR $BFB4
E033 A9 C4 LDA #C4
E035 A0 5F LDY #5F
E037 20 59 E0 JSR $E059
E03A A9 00 LDA #00
E03C B5 6F STA $6F
E03E 68 PLA
E03F 20 B9 BA JSR $BAB9
E042 60 RTS

```

Zeiger auf Konstante 1/LOG(2)  
mit FAC multiplizieren

Mantisse von FAC um eins erhöhen

FAC nach ARG bringen  
Exponent  
Zahl größer 128 ?

falls positiv 'overflow'  
INTEGER-Funktion

gleich 127 ?

FAC und ARG vertauschen

ARG - FAC  
Vorzeichenwechsel

Zeiger auf Polynomkoeffizienten  
Polynom berechnen

Exponenten von FAC und ARG addieren

\*\*\*\*\*

Polynomberechnung  $y=a1*x+a2*x^3+a3*x^5+...$ 

```

E043 85 71 STA $71
E045 B4 72 STY $72
E047 20 CA BB JSR $BBCA
E04A A9 57 LDA #57
E04C 20 28 BA JSR $BA28
E04F 20 5D E0 JSR $E05D
E052 A9 57 LDA #57
E054 A0 00 LDY #00
E056 4C 28 BA JMP $BA28

```

Zeiger auf Polynomkoeffizienten  
FAC nach Akku#3 bringen  
Zeiger auf Akku #3  
FAC \* Akku#3 (quadrieren)  
Polynomberechnung

Zeiger auf Akku#3  
FAC = FAC \* Akku#3

\*\*\*\*\*

Polynomberechnung  $y=a0+a1*x+a2*x^2+a3*x^3+...$ 

```

E059 85 71 STA $71
E05B B4 72 STY $72
E05D 20 C7 BB JSR $BBC7

```

Zeiger auf Polynomgrad  
FAC nach Akku#4 bringen

E060	B1 71	LDA (\$71),Y	Polynomgrad
E062	85 67	STA \$67	als Zahler
E064	A4 71	LDY \$71	
E066	C8	INY	
E067	98	TYA	Zeiger erhöhen, zeigt dann
E068	D0 02	BNE \$E06C	auf ersten Koeffizienten
E06A	E6 72	INC \$72	
E06C	B3 71	STA \$71	
E06E	A4 72	LDY \$72	
E070	20 28 BA	JSR \$BA28	FAC = FAC * Konstante (Zeiger A/Y)
E073	A5 71	LDA \$71	
E075	A4 72	LDY \$72	
E077	18	CLC	
E078	69 05	ADC #05	Zeiger um 5 erhöhen - nächste Zahl
E07A	90 01	BCC \$E07D	
E07C	C8	INY	
E07D	85 71	STA \$71	
E07F	84 72	STY \$72	
E081	20 67 BB	JSR \$BB67	FAC = FAC + Konstante (Zeiger A/Y)
E084	A9 5C	LDA #5C	
E086	A0 00	LDY #00	Zeiger auf Adresse 4
E088	C6 67	DEC \$67	Zähler erniedrigen
E08A	D0 E4	BNE \$E070	
E08C	60	RTS	

*****			Konstanten für RND
E08D	98 35 44 7A 00		11879546
E092	68 28 B1 46 00		3.92767774E-4

*****			BASIC-Funktion RND
E097	20 28 BC	JSR \$BC28	Vorzeichen holen
E09A	30 37	BMI \$E0D3	negativ ?
E09C	D0 20	BNE \$E0BE	
E09E	20 F3 FF	JSR \$FFF3	Basis-Adresse CIA holen
E0A1	86 22	STX \$22	
E0A3	B4 23	STY \$23	als Zeiger speichern
E0A5	A0 04	LDY #04	
E0A7	B1 22	LDA (\$22),Y	Timer A low
E0A9	B5 62	STA \$62	
E0AB	C8	INY	
E0AC	B1 22	LDA (\$22),Y	Timer A high
E0AE	B5 64	STA \$64	
E0B0	A0 00	LDY #00	
E0B2	B1 22	LDA (\$22),Y	TOD 1/10 sec
E0B4	B5 63	STA \$63	
E0B6	C8	INY	
E0B7	B1 22	LDA (\$22),Y	TOD sec
E0B9	B5 65	STA \$65	
E0BB	4C E3 E0	JMP \$E0E3	
E0BE	A9 BB	LDA #BB	
E0C0	A0 00	LDY #00	Zeiger auf letzten RND-Wert
E0C2	20 A2 BB	JSR \$BBA2	nach FAC holen
E0C5	A9 BD	LDA #BD	
E0C7	A0 E0	LDY #E0	Zeiger auf Konstante
E0C9	20 28 BA	JSR \$BA28	FAC = FAC * Konstante
E0CC	A9 92	LDA #92	
E0CE	A0 E0	LDY #E0	Zeiger auf Konstante
E0D0	20 67 BB	JSR \$BB67	FAC = FAC + Konstante
E0D3	A6 65	LDX #65	
E0D5	A5 62	LDA \$62	

E0D7	B5 65	STA \$65	
E0D9	B6 62	STX \$62	Stellen 1a FAC vertauschen
E0DB	A6 63	LDX \$63	
E0DD	A5 64	LDA \$64	
E0DF	B5 63	STA \$63	
E0E1	B6 64	STX \$64	
E0E3	A5 00	LDA \$800	
E0E5	B5 66	STA \$66	
E0E7	A5 61	LDA \$61	
E0E9	B5 70	STA \$70	
E0EB	A9 80	LDA \$8B0	Exponent
E0ED	B5 61	STA \$61	
E0EF	20 D7 BB	JSR \$8BD7	FAC linksbündig machen
E0F2	A2 BB	LDX \$8B	
E0F4	A0 00	LDY \$800	Zeiger auf letzten RND-Wert
E0F6	4C D4 BB	JMP \$8BD4	FAC runden und speichern
*****			Fehlerauswertung nach I/O-Routinen
E0F9	C9 F0	CMP \$F0	RS 232 OPEN oder CLOSE ?
E0FB	D0 07	BNE \$E104	nein
E0FD	B4 3B	STY \$3B	BASIC-RAM Ende neu setzen
E0FF	B6 37	STX \$37	
E101	4C 63 A6	JMP \$A663	und zum CLR-Befehl
E104	AA	TAX	Fehlernummer nach X
E105	D0 02	BNE \$E109	nicht Null ?
E107	A2 1E	LDX \$81E	sonst Nummer für 'break'
E109	4C 37 A4	JMP \$A437	Fehlermeldung ausgeben
*****			BASIC BSOUT
E10C	20 D2 FF	JSR \$FFD2	ein Zeichen ausgeben
E10F	B0 EB	BCS \$E0F9	Fehler ?
E111	60	RTS	
*****			BASIC BASIN
E112	20 CF FF	JSR \$FFCF	ein Zeichen holen
E115	B0 E2	BCS \$E0F9	Fehler ?
E117	60	RTS	
*****			BASIC CKOUT
E118	20 AD E4	JSR \$E4AD	Ausgabegerät setzen
E11B	B0 DC	BCS \$E0F9	Fehler ?
E11D	60	RTS	
*****			BASIC CHKIN
E11E	20 C6 FF	JSR \$FFC6	Eingabegerät setzen
E121	B0 D6	BCS \$E0F9	Fehler ?
E123	60	RTS	
*****			BASIC GETIN
E124	20 E4 FF	JSR \$FFE4	ein Zeichen holen
E127	B0 D0	BCS \$E0F9	Fehler ?
E129	60	RTS	
*****			SYS-Befehl
E12A	20 BA AD	JSR \$ADBA	FRMNUM, numerischen Ausdruck holen
E12D	20 F7 B7	JSR \$B7F7	in Adressformat wandlen, nach \$14/\$15
E130	A9 E1	LDA \$8E1	
E132	4B	PHA	Rucksprungadresse auf Stack
E133	A9 46	LDA \$846	
E135	4B	PHA	

E136	AD 0F 03	LDA #030F	Status,
E139	4B	PHA	
E13A	AD 0C 03	LDA #030C	Akku,
E13D	AE 0D 03	LDX #030D	X-Register und
E140	AC 0E 03	LDY #030E	Y-Register übergeben
E143	2B	PLP	Status setzen
E144	6C 14 00	JMP (#0014)	Routine aufrufen
E147	0B	PHP	Status speichern
E14B	BD 0C 03	STA #030C	Akku,
E14B	BE 0D 03	STX #030D	X-Register,
E14E	BC 0E 03	STY #030E	Y-Register und
E151	6B	PLA	
E152	BD 0F 03	STA #030F	Status wieder speichern
E155	60	RTS	
***** SAVE-Befehl			
E156	20 D4 E1	JSR \$E1D4	Parameter (Dateinamen, Prim. und Sek. Adresse)
E159	A6 2D	LDX #2D	Endadresse gleich BASIC-Programm Ende
E15B	A4 2E	LDY #2E	
E15D	A9 2B	LDA #2B	Startadresse gleich Zeiger auf BASIC Anfang
E15F	20 DB FF	JSR \$FFDB	Save-Routine
E162	B0 95	BCS #E0F9	Fehler ?
E164	60	RTS	
***** VERIFY-Befehl			
E165	A9 01	LDA #01	Verify-Flag
E167	2C	.BYTE #2C	
***** LOAD-Befehl			
E16B	A9 00	LDA #00	Load-Flag
E16A	B5 0A	STA #0A	speichern
E16C	20 D4 E1	JSR \$E1D4	Parameter holen
E16F	A5 0A	LDA #0A	Flag
E171	A6 2B	LDX #2B	Startadresse gleich BASIC-Start
E173	A4 2C	LDY #2C	
E175	20 D5 FF	JSR \$FFD5	Load-Routine
E17B	B0 57	BCS #E1D1	Fehler ?
E17A	A5 0A	LDA #0A	Load/Verify - Flag
E17C	F0 17	BEQ #E195	
E17E	A2 1C	LDX #1C	Offset für 'verify error'
E180	20 B7 FF	JSR \$FFB7	Status holen
E183	29 10	AND #10	Fehler-Bit isolieren
E185	D0 17	BNE #E19E	Statusbit gesetzt, dann Fehler
E187	A5 7A	LDA #7A	
E189	C9 02	CMP #02	Test auf Direkt-Modus
E18B	F0 07	BEQ #E194	ja, dann fertig
E18D	A9 64	LDA #64	Zeiger auf 'ok'
E18F	A0 A3	LDY #A3	
E191	4C 1E AB	JMP #AB1E	ausgeben
E194	60	RTS	
E195	20 B7 FF	JSR \$FFB7	Status holen
E19B	29 BF	AND #BF	EOF-Bit löschen
E19A	F0 05	BEQ #E1A1	kein Fehler
E19C	A2 1D	LDX #1D	Offset für 'load error'
E19E	4C 37 A4	JMP #A437	Fehlermeldung ausgeben
E1A1	A5 7B	LDA #7B	
E1A3	C9 02	CMP #02	Direkt-Modus testen
E1A5	D0 0E	BNE #E1B5	nein, dann weiter
E1A7	B6 2D	STX #2D	Endadresse gleich Programmende
E1A9	B4 2E	STY #2E	

E1A8	A9 76	LDA #876	Zeiger auf 'ready'
E1AD	A0 A3	LDY #8A3	
E1AF	20 1E AB	JSR \$AB1E	String ausgeben
E1B2	4C 2A A5	JMP \$A52A	Programmzeilen neu binden, CLR
E1B5	20 8E A6	JSR \$A68E	CHRGET-Zeiger auf Programmstart
E1B8	20 33 A5	JSR \$A533	Programmzeilen neu binden
E1BB	4C 77 A6	JMP \$A677	RESTORE, BASIC initialisieren
*****			BASIC-Befehl OPEN
E1BE	20 19 E2	JSR \$E219	Parameter holen
E1C1	20 C0 FF	JSR \$FFC0	OPEN-Routine
E1C4	80 0B	BCS \$E1D1	Fehler ?
E1C6	60	RTS	
*****			BASIC-Befehl CLOSE
E1C7	20 19 E2	JSR \$E219	Parameter holen
E1CA	A5 49	LDA \$49	Filenummer
E1CC	20 C3 FF	JSR \$FFC3	CLOSE-Routine
E1CF	90 C3	BCC \$E194	kein Fehler, RTS
E1D1	4C F9 E0	JMP \$E0F9	zur Fehlerauswertung
*****			Parameter für LOAD und SAVE holen
E1D4	A9 00	LDA #800	Default für Länge des Filenamens
E1D6	20 BD FF	JSR \$FFBD	Filenameparameter setzen
E1D9	A2 01	LDX #801	Default für Geratenummer
E1DB	A0 00	LDY #800	" " Sekundaradresse
E1DD	20 BA FF	JSR \$FFBA	Fileparameter setzen
E1E0	20 06 E2	JSR \$E206	weitere Zeichen ?
E1E3	20 57 E2	JSR \$E257	Filename holen
E1E6	20 06 E2	JSR \$E206	weitere Zeichen ?
E1E9	20 00 E2	JSR \$E200	Parameter holen
E1EC	A0 00	LDY #800	
E1EE	86 49	STX \$49	
E1F0	20 BA FF	JSR \$FFBA	Fileparameter setzen
E1F3	20 06 E2	JSR \$E206	weitere Zeichen ?
E1F6	20 00 E2	JSR \$E200	Parameter holen
E1F9	BA	TXA	
E1FA	AB	TAY	
E1FB	A6 49	LDX \$49	
E1FD	4C BA FF	JMP \$FFBA	Fileparameter setzen
*****			
E200	20 0E E2	JSR \$E20E	prüft auf Komma und weitere Zeichen
E203	4C 9E B7	JMP \$B79E	holt Byte-Wert nach X
*****			prüft auf weitere Zeichen
E206	20 79 00	JSR \$0079	CHRGET letztes Zeichen
E209	D0 02	BNE \$E20D	weitere Zeichen, dann Rückkehr
E20B	6B	PLA	sonst Rückkehr zur übergeordneten Routine
E20C	6B	PLA	
E20D	60	RTS	
*****			
E20E	20 FD AE	JSR \$AEFD	prüft auf Komma
E211	20 79 00	JSR \$0079	CHRGET letztes Zeichen holen
E214	D0 F7	BNE \$E20D	weitere Zeichen, dann Rückkehr
E216	4C 0B AF	JMP \$AF0B	SYNTAX ERROR
*****			Parameter für OPEN holen

E219	A9 00	LDA #100	Default für Länge des Filenamens
E21B	20 BD FF	JSR \$FFB0	Filenamenparameter setzen
E21E	20 11 E2	JSR \$E211	weitere Zeichen ?
E221	20 9E B7	JSR \$B79E	holt logische Filenummer nach 1
E224	B6 49	STX \$49	
E226	BA	TXA	
E227	A2 01	LDY #01	Default für Geräteadresse
E229	A0 00	LDY #00	" Sekundaradresse
E22B	20 BA FF	JSR \$FFBA	Fileparameter setzen
E22E	20 06 E2	JSR \$E206	weitere Zeichen ?
E231	20 00 E2	JSR \$E200	holt Geräteadresse
E234	B6 4A	STX \$4A	
E236	A0 00	LDY #00	
E238	A5 49	LDA \$49	
E23A	E0 03	CPX #03	
E23C	90 01	BCC \$E23F	
E23E	B8	DEY	
E23F	20 BA FF	JSR \$FFBA	Fileparameter setzen
E242	20 06 E2	JSR \$E206	weitere Zeichen ?
E245	20 00 E2	JSR \$E200	holt Sekundaradresse
E248	BA	TXA	
E249	AB	TAY	
E24A	A6 4A	LDX \$4A	
E24C	A5 49	LDA \$49	
E24E	20 BA FF	JSR \$FFBA	Fileparameter setzen
E251	20 06 E2	JSR \$E206	weitere Zeichen ?
E254	20 0E E2	JSR \$E20E	prüft auf Komma
E257	20 9E AD	JSR \$AD9E	FRNEVL Ausdruck holen
E25A	20 A3 B6	JSR \$B6A3	holt Stringparameter, FRESTR
E25D	A6 22	LDX \$22	
E25F	A4 23	LDY \$23	Adresse des Filenamens
E261	4C BD FF	JMP \$FFBD	Filenamenparameter setzen

\*\*\*\*\*

E264	A9 E0	LDA #E0	BASIC-Funktion COS
E266	A0 E2	LDY #E2	Zeiger auf Konstante Pi/2
E268	20 67 BB	JSR \$B067	zu FAC addieren

\*\*\*\*\*

E26B	20 0C BC	JSR \$BC0C	BASIC-Funktion SIN
E26E	A9 E5	LDA #E5	FAC runden und nach ARG
E270	A0 E2	LDY #E2	Zeiger auf Konstante Pi*2
E272	A6 6E	LDX \$6E	
E274	20 07 BB	JSR \$BB07	FAC durch 2*Pi dividieren
E277	20 0C BC	JSR \$BC0C	FAC runden und nach ARG
E27A	20 CC BC	JSR \$BCCC	INT - Funktion
E27D	A9 00	LDA #00	
E27F	B5 6F	STA \$6F	
E281	20 53 BB	JSR \$BB53	ARG minus FAC
E284	A9 EA	LDA #EA	
E286	A0 E2	LDY #E2	Zeiger auf Konstante 0.25
E288	20 50 BB	JSR \$BB50	0.25 - FAC
E28B	A5 66	LDA \$66	
E28D	4B	PHA	Vorzeichen auf Stack
E28E	10 0D	BPL \$E29D	positiv ?
E290	20 49 BB	JSR \$BB49	FAC + 0.5
E293	A5 66	LDA \$66	Vorzeichen
E295	30 09	BMI \$E2A0	negativ ?
E297	A5 12	LDA \$12	
E299	49 FF	EOR #FF	Flag umdrehen

E29B	05 12	STA \$12	
E29D	20 B4 BF	JSR \$BFB4	Vorzeichen wechseln
E2A0	A9 EA	LDA #\$EA	
E2A2	A0 E2	LDY #\$E2	Zeiger auf Konstante 0.25
E2A4	20 67 B8	JSR \$B867	FAC ← 0.25
E2A7	68	PLA	Vorzeichen holen
E2A8	10 03	BPL \$E2AD	positiv ?
E2AA	20 B4 BF	JSR \$BFB4	Vorzeichen wechseln
E2AD	A9 EF	LDA #\$EF	
E2AF	A0 E2	LDY #\$E2	Zeiger auf Polynomkoeffizienten
E2B1	4C 43 E0	JMP \$E043	Polynom berechnen

*****			BASIC-Funktion TAN
E2B4	20 CA BB	JSR \$BBCA	FAC nach Akku#3
E2B7	A9 00	LDA #\$00	
E2B9	05 12	STA \$12	Flag setzen
E2BB	20 6B E2	JSR \$E26B	SIN berechnen
E2BE	A2 4E	LDX #\$4E	
E2C0	A0 00	LDY #\$00	Zeiger auf Hilfsakku
E2C2	20 F6 E0	JSR \$E0F6	FAC nach Hilfsakku
E2C5	A9 57	LDA #\$57	
E2C7	A0 00	LDY #\$00	Zeiger auf Akku#3
E2C9	20 A2 BF	JSR \$BBA2	Akku#3 nach FAC
E2CC	A9 00	LDA #\$00	
E2CE	05 66	STA \$66	Vorzeichen
E2D0	A5 12	LDA \$12	Flag
E2D2	20 DC E2	JSR \$E2DC	COS berechnen
E2D5	A9 4E	LDA #\$4E	
E2D7	A0 00	LDY #\$00	Zeiger auf Hilfsakku (SIN)
E2D9	4C 0F BB	JMP \$BB0F	durch FAC dividieren
E2DC	4B	PHA	
E2DD	4C 9D E2	JMP \$E29D	COS berechnen

*****			Konstanten für SIN und COS
E2E0	B1 49 0F DA A2		1.57079633    Pi/2
E2E5	B3 49 0F DA A2		6.28318531    2*Pi
E2EA	7F 00 00 00 00		.25
E2EF	05		S = Polynomgrad, 6 Koeffizienten
E2F0	B4 E6 1A 2D 1B		-14.3813907
E2F5	B6 2B 07 FB F8		42.0077971
E2FA	B7 99 6B B9 01		-76.7041703
E2FF	B7 23 35 DF E1		81.6052237
E304	B6 A5 5D E7 2B		-41.3147021
E309	B3 49 0F DA A2		6.28318531    2*Pi

*****			BASIC-Funktion ATN
E30E	A5 66	LDA \$66	Vorzeichen
E30F	4B	PHA	retten
E311	10 03	BPL \$E316	positiv ?
E313	20 B4 BF	JSR \$BFB4	Vorzeichen vertauschen
E316	A5 61	LDA \$61	Exponent
E31B	4B	PHA	retten
E319	C9 81	CMP #\$81	Zahl mit 1 vergleichen
E31B	90 07	BCC \$E324	kleiner ?
E31D	A9 BC	LDA #\$BC	
E31F	A0 B9	LDY #\$B9	Zeiger auf Konstante 1
E321	20 0F BB	JSR \$BB0F	1 durch FAC dividieren (Kehrwert)
E324	A9 3E	LDA #\$3E	
E326	A0 E3	LDY #\$E3	Zeiger auf Polynomkoeffizienten
E32B	20 43 E0	JSR \$E043	Polynom berechnen



E32B	6B	PLA	Exponent zuruckholen
E32C	C9 B1	CMP #81	war Zahl kleiner 1 ?
E32E	90 07	BCC #E337	
E330	A9 E0	LDA #E0	
E332	A0 E2	LDY #E2	Zeiger auf Konstante P1/2
E334	20 50 88	JSR \$B850	P1/2 minus FAC
E337	68	PLA	Vorzeichen holen
E338	10 03	BPL #E33D	positiv ?
E33A	4C B4 BF	JMP \$BFB4	Vorzeichen wechseln
E33D	60	RTS	
*****			
E33E	0B		Fließkommakonstanten für ATN-Funktion
E33F	76 B3 B3 BD D3		11 = Polynomgrad, dann 12 Koeffizienten
E344	79 1E F4 A6 F5		-6.84793912E-04
E349	7B B3 FC B0 10		4.85094216E-03
E34E	7C 0C 1F 67 CA		-.0161117015
E353	7C DE 53 CB C1		.034209638
E356	7D 14 64 70 4C		-.054279133
E35C	7D B7 EA 51 7A		.0724571965
E362	7D 63 30 8B 7E		-.0898019185
E367	7E 92 44 99 3A		.110932413
E36C	7E 4C CC 91 C7		-.142839808
E371	7F AA AA AA 13		.199999912
E376	B1 00 00 00 00		-.333333316
			1
*****			
E378	20 CC FF	JSR \$FFCC	BASIC NMI-Einsprung
E37E	A9 00	LDA #00	CLRCH
E380	B5 13	STA #13	
E382	20 7A A6	JSR \$A67A	Eingabegerat gleich Tastatur
E385	5B	CLI	BASIC initialisieren
E386	A2 80	LDX #80	
E388	6C 00 03	JMP (#0300)	Flag für kein Fehler
E38B	8A	TXA	BASIC Warastart Vektor JMP #E38B
E38C	30 03	BMI #E391	Fehlernummer in Akku
E38E	4C 3A A4	JMP \$A43A	kein Fehler, dann 'ready.'
E391	4C 74 A4	JMP \$A474	Fehlermeldung ausgeben
			Ready - Modus
*****			
E394	20 53 E4	JSR #E453	BASIC Kaltstart
E397	20 BF E3	JSR #E3BF	BASIC-Vektoren setzen
E39A	20 22 E4	JSR #E422	RAM initialisieren
E39D	A2 FB	LDX #FB	Einschaltmeldung ausgeben
E39F	9A	TXS	Stackpointer setzen
E3A0	D0 E4	BNE #E3B6	
			zum Warastart
*****			
E3A2	E6 7A	INC #7A	Kopie der CHRGET-Routine
E3A4	D0 02	BNE #E3AB	
E3A6	E6 7B	INC #7B	Zeiger in BASIC-Text erhöhen
E3AB	AD 60 EA	LDA #EA60	
E3AB	C9 3A	CMP #3A	' ; '
E3AD	B0 0A	BCS #E3B9	
E3AF	C9 20	CMP #20	' ' Leerzeichen überlesen
E3B1	F0 EF	BEQ #E3A2	
E3B3	3B	SEC	
E3B4	E9 30	SBC #30	Test auf Ziffer, dann C=1
E3B6	3B	SEC	
E3B7	E9 D0	SBC #D0	

E3B9 50 RTS

\*\*\*\*\* Anfangswert für RND-Funktion  
E3BA B0 4F C7 52 5B .611635157

\*\*\*\*\* RAM für BASIC initialisieren  
E3BF A9 4C LDA #4C JMP  
E3C1 B5 54 STA #54 für Funktionen  
E3C3 B0 10 03 STA #0310 für USR-Funktion  
E3C6 A9 4B LDA #4B Zeiger auf 'illegal quantity'  
E3C8 A0 B2 LDY #B2  
E3CA BD 11 03 STA #0311 als USR-Vektor speichern  
E3CD BC 12 03 STY #0312  
E3D0 A9 91 LDA #91 #B391  
E3D2 A0 B3 LDY #B3  
E3D4 B5 05 STA #05 als Vektor für Fest/Fließkomma-Wandlung  
E3D6 B4 06 STY #06  
E3D8 A9 AA LDA #AA #B1AA  
E3DA A0 B1 LDY #B1  
E3DC B5 03 STA #03 als Vektor für Fließ/Festkomma-Wandlung  
E3DE B4 04 STY #04  
E3E0 A2 1C LDX #1C  
E3E2 BD 42 E3 LDA #E3A2.X CHRGET-Routine  
E3E5 95 73 STA #73.X ins RAM kopieren  
E3E7 CA DEX  
E3E8 10 F8 BPL #E3E2  
E3EA A9 03 LDA #03  
E3EC B5 53 STA #53 Schrittweite für garbage collect  
E3EE A9 00 LDA #00  
E3F0 B5 6B STA #6B  
E3F2 B5 13 STA #13 Eingabegerät gleich Tastatur  
E3F4 B5 1B STA #1B  
E3F6 A2 01 LDX #01  
E3F8 BE FD 01 STX #01FD  
E3FB BE FC 01 STX #01FC  
E3FE A2 19 LDX #19  
E400 B6 16 STY #16 Zeiger für Stringverwaltung  
E402 3B SEC  
E403 20 9C FF JSR #FF9C RAM Start holen  
E406 B6 2B STX #2B  
E408 B4 2C STY #2C als BASIC-Start speichern  
E40A 3B SEC  
E40B 20 99 FF JSR #FF99 RAM Ende holen  
E40E B6 37 STX #37  
E410 B4 3B STY #3B als BASIC-Ende speichern  
E412 B6 33 STX #33  
E414 B4 34 STY #34  
E416 A0 00 LDY #00  
E418 9B TYA  
E419 91 2B STA (#2B),Y #00 an BASIC-Start  
E41B E6 2B INC #2B  
E41D D0 02 BNE #E421 BASIC-Start + 1  
E41F E6 2C INC #2C  
E421 60 RTS

\*\*\*\*\*  
E422 A5 2B LDA #2B  
E424 A4 2C LDY #2C Zeiger auf BASIC-RAM Start  
E426 20 0B A4 JSR #A40B prüft auf Platz im Speicher  
E429 A9 73 LDA #73 Zeiger auf Einschaltmeldung

```

E42B A0 E4 LDY #E4
E42D 20 1E AB JSR #AB1E String ausgeben
E430 A5 37 LDA #37
E432 38 SEC BASIC-Ende
E433 E5 2B SBC #2B
E435 AA TAX minus BASIC-Start
E436 A5 38 LDA #38
E438 E5 2C SBC #2C gleich Bytes free
E43A 20 CD BD JSR #BDCD Anzahl ausgeben
E43D A9 60 LDA #60 Zeiger auf 'basic bytes free'
E43F A0 E4 LDY #E4
E441 20 1E AB JSR #AB1E String ausgeben
E444 4C 44 A6 JMP #A644 zum NEW-Befehl

```

\*\*\*\*\* Tabelle der BASIC-Vektoren

```

E447 BB E3 B3 A4 7C A5 1A A7
E44F E4 A7 B6 AE

```

\*\*\*\*\* BASIC-Vektoren laden

```

E453 A2 0B LDX #0B
E455 BD 47 E4 LDA #E447,X
E458 9D 00 03 STA #0300,X
E45B CA DEX
E45C 10 F7 BPL #E453
E45E 60 RTS

```

\*\*\*\*\* Betriebssystem

\*\*\*\*\* System-Meldungen

```

E45F 00 20 42 41 53 49 43 20 basic bytes free
E467 42 59 54 43 53 20 46 52
E46F 45 43 0D 00
E473 93 0D 20 20 20 20 2A 2A (clr) *** commodore 64 basic v2 ***
E47B 2A 2A 20 43 4F 4D 4D 4F (cr) (cr) 64k ram system
E483 44 4F 52 43 20 36 34 20
E48B 42 41 53 49 43 20 56 32
E493 20 2A 2A 2A 2A 0D 0D 20
E49B 36 34 4B 20 52 41 4D 20
E4A3 53 59 53 54 43 4D 20 20
E4AB 00

```

```

E4AC 5C

```

\*\*\*\*\* BASIC-CKOUT Routine

```

E4AD 4B PHA
E4AE 20 C9 FF JSR #FFC9 CKOUT Ausgabegerät setzen
E4B1 AA TAX Fehlernummer nach X
E4B2 6B PLA
E4B3 90 01 BCC #E4B6 kein Fehler ?
E4B5 BA TXA Fehlernummer wieder in Akku
E4B6 60 RTS

```

```

E4B7 AA .....
E4D9 ..... AA

```

\*\*\*\*\* Hintergrundfarbe setzen

```

E4DA AD 21 D0 LDA #D021 Farbe holen
E4DD 91 F3 STA ($F3),Y ins Farbram schreiben
E4DF 60 RTS

```

```

***** wartet auf Commodore-Taste
E4E0 69 02 ADC #$02 2*256/60 = 8.5 Sekunden warten
E4E2 A4 91 LDY $91 Flag testen
E4E4 C8 INY
E4E5 D0 04 BNE $E4EB Taste gedrückt ?
E4E7 C5 A1 CMP $A1 Zeit noch nicht um ?
E4E9 D0 F7 BNE $E4E2
E4EB 60 RTS

***** Timerkonstanten für RS 232 Baud Rate, PAL-Version
E4EC 19 26 $2619 = 9753 50 Baud
E4EE 44 19 $1944 = 6468 75 Baud
E4F0 1A 11 $111A = 4378 110 Baud
E4F2 EB 0D $0DEB = 3560 134.5 Baud
E4F4 70 0C $0C70 = 3184 150 Baud
E4F6 06 06 $0606 = 1542 300 Baud
E4F8 D1 02 $02D1 = 736 600 Baud
E4FA 37 01 $0137 = 311 1200 Baud
E4FC AE 00 $00AE = 174 1800 Baud
E4FE 69 00 $0069 = 105 2400 Baud

***** Basis-Adresse des CIAs holen
E500 A2 00 LDY #$00
E502 A0 DC LDY #$DC $DC00
E504 60 RTS

***** holt Anzahl der Zeilen und Spalten
E505 A2 2B LDY #$2B 40 Spalten
E507 A0 19 LDY #$19 25 Zeilen
E509 60 RTS

***** Cursor setzen (C=0) / holen (C=1)
E50A B0 07 BCS $E513
E50C 86 D6 STX $D6 Zeile
E50E B4 D3 STY $D3 Spalte
E510 20 6C E5 JSR $E56C Cursor setzen
E513 A6 D6 LDX $D6
E515 A4 D3 LDY $D3
E517 60 RTS

***** Bildschirm Reset
E518 20 A0 E5 JSR $E5A0 Videocontroller initialisieren
E51B A9 00 LDA #$00
E51D BD 91 02 STA $0291 Shift-Commodore ermöglichen
E520 B5 CF STA $CF Cursor nicht in Blinkphase
E522 A9 4B LDA #$4B
E524 BD BF 02 STA $02BF ($02BF) = $EB4B
E527 A9 EB LDA #$EB
E529 BD 90 02 STA $0290 = Zeiger auf Adressen für Tastaturdekodierung
E52C A9 0A LDA #$0A 10
E52E B0 89 02 STA $0289 max. Länge des Tastaturpuffers
E531 BD 8C 02 STA $028C Zähler für Repeat-Geschwindigkeit
E534 A9 0E LDA #$0E hellblau
E536 BD 86 02 STA $0286 Augenblickliche Farbe
E539 A9 04 LDA #$04
E53B BD 8B 02 STA $028B Repeat-Geschwindigkeit
E53E A9 0C LDA #$0C
E540 B5 CD STA $CD Cursor Blinkzeit
E542 B5 CC STA $CC Cursor Blinkflag

```

```

***** Bildschirm löschen
E544 AD 88 02 LDA #0288 Speicherseite für Bildschirm-RAM
E547 09 80 ORA #180
E549 AB TAY
E54A A9 00 LDA #100
E54C AA TAX
E54D 94 D9 STY $D9,X Adressen der Bildschirmzeilen
E54F 18 CLC
E550 69 28 ADC #128 40 addieren (eine Zeile)
E552 90 01 BCC $E555
E554 C8 INY
E555 E8 INX
E556 E0 1A CPX #1A 26. alle Zeilen ?
E558 D0 F3 BNE $E54D
E55A A9 FF LDA #FF
E55C 95 D9 STA $D9,X
E55E A2 18 LDY #18 24, Anzahl der Zeilen minus 1
E560 20 FF E9 JSR $E9FF Bildschirmzeile löschen
E563 CA DEX
E564 10 FA BPL $E560

***** Cursor Home
E566 A0 00 LDY #00
E568 84 D3 STY $D3 Cursorspalte
E56A 84 D6 STY $D6 Cursorzeile

***** Cursorpos. berechnen, Bildschirmzeiger setzen
E56C A6 D6 LDX $D6 Cursorzeile
E56E A5 D3 LDA $D3 Cursorspalte
E570 84 D9 LDY $D9,X
E572 30 0B BMI $E57C
E574 18 CLC
E575 69 28 ADC #128 +40
E577 85 D3 STA $D3
E579 CA DEX
E57A 10 F4 BPL $E570
E57C 85 D9 LDA $D9,X MSB der Startadresse der Zeile
E57E 29 03 AND #03
E580 0D 88 02 ORA #0288 Video-Ram Page (hi Byte)
E583 85 D2 STA $D2
E585 8D F0 EC LDA $ECF0,X LSB der Startadresse der Zeile
E588 85 D1 STA $D1 als Zeiger nach $D1/$D2
E58A A9 27 LDA #27 39
E58C E8 INY
E58D 84 D9 LDY $D9,X
E58F 30 0B BMI $E597
E591 18 CLC
E592 69 28 ADC #128 +40
E594 E8 INX
E595 10 F6 BPL $E58D
E597 85 D3 STA $D3
E599 60 RTS

*****
E59A 20 A0 E5 JSR $E5A0 Videocontroller initialisieren
E59D 4C 66 E5 JMP $E566 Cursor Home

***** Videocontroller initialisieren
E5A0 A9 03 LDA #03
E5A2 85 9A STA $9A Ausgabe auf Bildschirm

```

E5A4	A9 00	LDA #000	
E5A6	B5 99	STA \$99	Eingabe von Tastatur
E5A8	A2 2F	LDX #02F	47
E5AA	8D B8 EC	LDA \$ECB8,X	Konstanten
E5AD	9D FF CF	STA \$CFFF,X	in Videokontroller schreiben
E5B0	CA	DEX	
E5B1	D0 F7	BNE \$E5AA	
E5B3	60	RTS	
***** Zeichen aus Tastaturpuffer holen			
E5B4	AC 77 02	LDY \$0277	erstes Zeichen holen
E5B7	A2 00	LDX #000	
E5B9	B0 78 02	LDA \$0278,X	Puffer nach vorne aufrücken
E5BC	9D 77 02	STA \$0277,X	
E5BF	E8	INX	
E5C0	E4 C6	CPX \$C6	mit Anzahl der Zeichen vergleichen
E5C2	D0 F5	BNE \$E5B9	
E5C4	C6 C6	DEC \$C6	Zeichenzahl erniedrigen
E5C6	98	TYA	Zeichen in Akku holen
E5C7	58	CLI	
E5C8	18	CLC	
E5C9	60	RTS	
***** Warteschleife für Tastatureingabe			
ESCA	20 16 E7	JSR \$E716	Zeichen auf Bildschirm ausgeben
ESCD	A5 C6	LDA \$C6	Anzahl der gedruckten Tasten
ESCF	B5 CC	STA \$CC	
ESD1	8D 92 02	STA \$0292	
ESD4	F0 F7	BEQ \$E5CD	keine Taste gedruckt, dann warten
ESD6	78	SEI	
ESD7	A5 CF	LDA \$CF	Cursor in Blink-Phase ?
ESD9	F0 0C	BEQ \$E5E7	nein
ESDB	A5 CE	LDA \$CE	Zeichen unter dem Cursor
ESDD	AE 87 02	LDX \$02B7	Farbe unter dem Cursor
ESE0	A0 00	LDY #000	
ESE2	B4 CF	STY \$CF	Cursor nicht in Blinkphase
ESE4	20 13 E4	JSR \$EA13	Zeichen und Farbe setzen
ESE7	20 B4 E5	JSR \$E5B4	Zeichen aus Tastaturpuffer holen
ESEA	C9 B3	CMP #0B3	Kode für 'Shift RUN' ?
ESEC	D0 10	BNE \$E5FE	
ESEE	A2 09	LDX #009	9 Zeichen
ESF0	78	SEI	
ESF1	B6 C6	STX \$C6	Zeichenzahl merken
ESF3	BD E6 EC	LDA \$ECE6,X	'load (cr) run (cr)'
ESF6	9D 76 02	STA \$0276,X	in Tastaturpuffer holen
ESF9	CA	DEX	
ESFA	D0 F7	BNE \$E5F3	
ESFC	F0 CF	BEQ \$E5CD	und auswerten
ESFE	C9 0D	CMP #0D	'CR'
E600	D0 C8	BNE \$E5CA	nein, dann zurück zur Warteschleife
E602	A4 D5	LDY #D5	Länge der Bildschirmzeile
E604	B4 D0	STY #D0	CR-Flag setzen
E606	B1 D1	LDA (\$D1),Y	Zeichen vom Bildschirm holen
E608	C9 20	CMP #20	Leerzeichen am Ende der Zeile eliminieren
E60A	D0 03	BNE \$E60F	
E60C	B8	DEY	
E60D	D0 F7	BNE \$E606	
E60F	C8	INX	
E610	B4 C8	STY \$C8	Position als Index merken
E612	A0 00	LDY #000	

E614	8C 92 02	STY \$0292	Cursorspalte gleich Null
E617	84 D3	STY \$D3	
E619	84 D4	STY \$D4	Hochkommaflag löschen
E61B	A5 C9	LDA \$C9	
E61D	30 1B	BMI \$E63A	
E61F	A6 D6	LDX \$D6	
E621	20 ED E6	JSR \$E6ED	
E624	E4 C9	CPY \$C9	
E626	D0 12	BNE \$E63A	
E62B	A5 CA	LDA \$CA	letzte Spalte
E62A	85 D3	STA \$D3	in Spaltenzeiger bringen
E62C	C5 C8	CMP \$C8	mit Index vergleichen
E62E	90 0A	BCC \$E63A	
E630	B0 2B	BCS \$E65D	
***** Ein Zeichen von Bildschirm holen			
E632	9B	TXA	
E633	4B	PHA	Register retten
E634	8A	TXA	
E635	4B	PHA	
E636	A5 D0	LDA \$D0	CR-Flag
E63B	F0 93	BEQ \$E5CD	nein, dann zur Warteschleife
E63A	A4 D3	LDY \$D3	Spalte
E63C	B1 D1	LDA (\$D1),Y	Zeichen vom Bildschirm holen
E63E	85 D7	STA \$D7	
E640	29 3F	AND #\$3F	
E642	06 D7	ASL \$D7	und nach ASCII wandeln
E644	24 D7	BIT \$D7	
E646	10 02	BPL \$E64A	
E64B	09 80	ORA #\$80	
E64A	90 04	BCC \$E650	
E64C	A6 D4	LDX \$D4	
E64E	D0 04	BNE \$E654	
E650	70 02	BVS \$E654	
E652	09 40	ORA #\$40	
E654	E6 D3	INC \$D3	Cursor eins weiter setzen
E656	20 84 E6	JSR \$E6B4	auf Hochkomma testen
E659	C4 C8	CPY \$C8	Cursor in letzter Spalte ?
E65B	D0 17	BNE \$E674	
E65D	A9 00	LDA #00	
E65F	B5 D0	STA \$D0	'CR'-Flag
E661	A9 0D	LDA #0D	
E663	A6 99	LDX #99	
E665	E0 03	CPX #03	Eingabe vom Bildschirm ?
E667	F0 06	BEQ \$E66F	ja
E669	A6 9A	LDX #9A	Ausgabe auf Bildschirm
E66B	E0 03	CPX #03	ja
E66D	F0 03	BEQ \$E672	
E66F	20 16 E7	JSR \$E716	Zeichen auf Bildschirm schreiben
E672	A9 0D	LDA #0D	
E674	85 D7	STA \$D7	
E676	6B	PLA	
E677	AA	TAX	Register zuruckholen
E67B	6B	PLA	
E679	AB	TAY	
E67A	A5 D7	LDA \$D7	Bildschirm-Kode
E67C	C9 DE	CMP #\$DE	mit Kode für Pi vergleichen
E67E	D0 02	BNE \$E6B2	
E680	A9 FF	LDA #FF	ja, durch BASIC-Kode für Pi ersetzen
E682	1B	CLC	

E683 60 RTS

```
***** auf Hochloona testen
E684 C9 22 CMP #22      "" ?
E686 D0 09 BNE $E690    nein, dann fertig
E688 A5 D4 LDA #D4
E68A 49 01 EOR #01      Hochloona-Flag undrehen
E68C 85 D4 STA #D4
E68E A9 22 LDA #22      Hochloona-Mode wieder herstellen
E690 60 RTS
```

```
***** Zeichen auf Bildschirm ausgeben
E691 09 40 ORA #40
E693 A6 C7 LDX #C7      RTS ~
E695 F0 02 BEQ $E69F    Umwandlung in Bildschirmcode
E697 09 80 ORA #80      ja, dann Bit 7 setzen
E699 A6 D8 LDX #D8
E69B F0 02 BEQ $E69F
E69D C6 D8 DEC #D8
E69F AE 86 02 LDX #0286  Farbcode
E6A2 20 13 E4 JSR #E413  Zeichen in Bildschirm-RAM schreiben
E6A5 20 B6 E6 JSR #E6B6  Tabelle der Zeilenanfänge aktualisieren
E6A8 68 PLA
E6A9 A8 TAY
E6AA A5 D8 LDA #D8
E6AC F0 02 BEQ $E6B0
E6AE 46 D4 LSR #D4      Hochloona-Modus löschen
E6B0 68 PLA
E6B1 AA TAX
E6B2 68 PLA
E6B3 18 CLC
E6B4 58 CLI
E6B5 60 RTS
```

```
***** MSB für Zeilenanfänge neu berechnen
E6B6 20 B3 E8 JSR #E6B3
E6B9 E6 D3 INC #D3
E6BB A5 D5 LDA #D5
E6BD C5 D3 CMP #D3
E6BF B0 3F BCS $E700
E6C1 C9 4F CMP #4F      79 Zeichen (Doppelzeile) ?
E6C3 F0 32 BEQ $E6F7
E6C5 AD 92 02 LDA #0292
E6C8 F0 03 BEQ $E6CD
E6CA 4C 67 E9 JMP $E967
E6CD A6 D6 LDX #D6      Zeile
E6CF E0 19 CPX #19      25 ?
E6D1 90 07 BCC $E6DA
E6D3 20 EA E8 JSR #E6EA
E6D6 C6 D6 DEC #D6
E6D8 A6 D6 LDX #D6
E6DA 16 D9 ASL #D9,X
E6DC 56 D9 LSR #D9,X
E6DE E8 INX
E6DF B5 D9 LDA #D9,X
E6E1 09 80 ORA #80
E6E3 95 D9 STA #D9,X
E6E5 CA DEX
E6E6 A5 D5 LDA #D5
E6E8 18 CLC
```



E6E9	69 28	ADC #128	40 addieren
E6EB	85 D5	STA #D5	
E6ED	B5 D9	LDA #D9,X	
E6EF	30 03	BMI #E6F4	
E6F1	CA	DEX	
E6F2	D0 F9	BNE #E6ED	
E6F4	4C F0 E9	JMP #E9F0	Zeiger auf Farbran für Zeile x
E6F7	C6 D6	DEC #D6	Zeile erniedrigen
E6F9	20 7C E8	JSR #E87C	
E6FC	A9 00	LDA #00	
E6FE	85 D3	STA #D3	Spalte = 0
E700	60	RTS	
*****			Ruckschritt in vorhergehende Zeile
E701	A6 D6	LDX #D6	Cursorzeile
E703	D0 06	BNE #E708	null ?
E705	B6 D3	STX #D3	Cursorspalte
E707	68	PLA	
E708	68	PLA	
E709	D0 9D	BNE #E6A8	
E70B	CA	DEX	
E70C	86 D6	STX #D6	Zeilennummer erniedrigen
E70E	20 6C E5	JSR #E56C	Cursorposition berechnen
E711	A4 D5	LDY #D5	
E713	84 D3	STY #D3	
E715	60	RTS	
*****			Ausgabe auf Bildschirm
E716	48	PHA	
E717	85 27	STA #D7	Zeichen merken
E719	8A	TXA	
E71A	48	PHA	Register retten
E71B	98	TYA	
E71C	48	PHA	
E71D	A9 00	LDA #00	
E71F	85 D0	STA #D0	
E721	A4 D3	LDY #D3	
E723	A5 D7	LDA #D7	
E725	10 03	BPL #E72A	
E727	4C D4 E7	JMP #E7D4	Zeichen größer \$7F behandeln
E72A	C9 0D	CMR #0D	'carriage return' ?
E72C	D0 03	BNE #E731	
E72E	4C 91 E8	JMP #E891	Return ausgeben
E731	C9 20	CMR #20	' '
E733	90 10	BCC #E745	druckendes Zeichen ?
E735	C9 60	CMR #60	
E737	90 04	BCC #E73D	
E739	29 DF	AND #DF	
E73B	D0 02	BNE #E73F	
E73D	29 3F	AND #3F	
E73F	20 84 E6	JSR #E684	Test auf Hochkomma
E742	4C 93 E6	JMP #E693	zur Ausgabe, ASCII-Kode in Bildschirmcode
E745	A6 D8	LDX #D8	
E747	F0 03	BEQ #E74C	
E749	4C 97 E6	JMP #E697	
E74C	C9 14	CMR #14	'DEL' ?
E74E	D0 2E	BNE #E77E	
E750	98	TYA	
E751	D0 06	BNE #E759	
E753	20 01 E7	JSR #E701	zurück in vorherige Zeile

E756	4C 73 E7	JMP \$E773	
E759	20 A1 E8	JSR \$E8A1	
E75C	88	DEY	
E75D	84 D3	STY \$D3	
E75F	20 24 EA	JSR \$EA24	Zeiger auf Farbran berechnen
E762	C8	INY	
E763	B1 D1	LDA (\$D1),Y	Zeichen vom Bildschirm
E765	88	DEY	
E766	91 D1	STA (\$D1),Y	eins nach links schieben
E768	C8	INY	
E769	B1 F3	LDA (\$F3),Y	Farbe
E76B	88	DEY	
E76C	91 F3	STA (\$F3),Y	eins nach links schieben
E76E	C8	INY	
E76F	C4 D5	CPY \$D5	
E771	D0 EF	BNE \$E762	
E773	A9 20	LDA #20	Blank einfügen
E775	91 D1	STA (\$D1),Y	
E777	AD B6 02	LDA \$02B6	
E77A	91 F3	STA (\$F3),Y	Farbcode setzen
E77C	10 4D	BPL \$E7CB	fertig
E77E	A6 D4	LDX \$D4	Hochkoma-Modus ?
E780	F0 03	BEQ \$E785	nein
E782	4C 97 E6	JMP \$E697	Zeichen revers ausgeben
E785	C9 12	CMR #12	'RVS ON' ?
E787	D0 02	BNE \$E788	nein
E789	B5 C7	STA \$C7	Flag für RVS setzen
E78B	C9 13	CMR #13	'HOME' ?
E78D	D0 03	BNE \$E792	nein
E78F	20 66 E5	JSR \$E566	ja, Cursor Home
E792	C9 1D	CMR #1D	'Cursor right' ?
E794	D0 17	BNE \$E7AD	nein
E796	C8	INY	
E797	20 B3 E8	JSR \$E8B3	
E79A	84 D3	STY \$D3	
E79C	88	DEY	
E79D	C4 D5	CPY \$D5	
E79F	90 09	BCC \$E7AA	
E7A1	C6 D6	DEC \$D6	
E7A3	20 7C E8	JSR \$E87C	
E7A6	A0 00	LDY #00	
E7A8	B4 D3	STY \$D3	Spalte gleich null
E7AA	4C A8 E6	JMP \$E6A8	fertig
E7AD	C9 11	CMR #11	'Cursor down' ?
E7AF	D0 1D	BNE \$E7CE	nein
E7B1	18	CLC	
E7B2	98	TYA	
E7B3	69 28	ADC #28	plus 40, eine Zeile tiefer
E7B5	A8	TAY	
E7B6	E6 D6	INC \$D6	
E7B8	C5 D5	CMR \$D5	
E7BA	90 EC	BCC \$E7AB	
E7BC	F0 EA	BEQ \$E7AB	
E7BE	C6 D6	DEC \$D6	
E7C0	E9 28	SBC #28	40 abziehen
E7C2	90 04	BCC \$E7CB	
E7C4	B5 D3	STA \$D3	
E7C6	D0 F8	BNE \$E7C0	
E7C8	20 7C E8	JSR \$E87C	
E7CB	4C A8 E6	JMP \$E6A8	fertig

E7CE	20 C8 E8	JSR \$E8CB	prüft auf Farbcodes
E7D1	4C 44 EC	JMP \$EC44	test auf weitere Sonderzeichen
*****			Zeichen größer \$127
E7D4	29 7F	AND #\$7F	Kode größer 127, Bit 7 löschen
E7D6	C9 7F	CMP #\$7F	'P1' ?
E7D8	D0 02	BNE \$E7DC	
E7DA	A9 5E	LDA #\$5E	Bildschirmcode für P1
E7DC	C9 20	CMP #\$20	Steuerzeichen ?
E7DE	90 03	BCC \$E7E3	ja
E7E0	4C 91 E6	JMP \$E691	druckendes Zeichen ausgeben
E7E3	C9 0D	CMP #\$0D	'Shift return' ?
E7E5	D0 03	BNE \$E7EA	
E7E7	4C 91 E8	JMP \$E891	neuen Zeile
E7EA	A6 D4	LDX \$D4	Hochroma-Modus ?
E7EC	D0 3F	BNE \$E82D	ja, Steuerzeichen revers ausgeben
E7EE	C9 14	CMP #\$14	'INS' ?
E7F0	D0 37	BNE \$E829	
E7F2	A4 D5	LDY \$D5	Zeilenlänge
E7F4	B1 D1	LDA (\$D1),Y	letztes Zeichen in Zeile
E7F6	C9 20	CMP #\$20	gleich Leerzeichen ?
E7F8	D0 04	BNE \$E7FE	
E7FA	C4 D3	CPY \$D3	Cursor in letzter Spalte ?
E7FC	D0 07	BNE \$E805	
E7FE	C0 4F	CPY #\$4F	79 ? maximale Zeilenlänge
E800	F0 24	BEQ \$E826	letzte Spalte, dann keine Aktion
E802	20 65 E9	JSR \$E965	Leerzeile einfügen
E805	A4 D5	LDY \$D5	Zeilenlänge
E807	20 24 EA	JSR \$EA24	Zeiger auf Farbraum berechnen
E80A	88	DEY	
E80B	B1 D1	LDA (\$D1),Y	Zeichen vom Bildschirm
E80D	C8	INY	
E80E	91 D1	STA (\$D1),Y	eins nach rechts schieben
E810	88	DEY	
E811	B1 F3	LDA (\$F3),Y	und Farbe
E813	C8	INY	
E814	91 F3	STA (\$F3),Y	verschieben
E816	88	DEY	
E817	C4 D3	CPY \$D3	bis zur aktuellen Position aufrücken
E819	D0 EF	BNE \$E80A	
E81B	A9 20	LDA #\$20	Leerzeichen
E81D	91 D1	STA (\$D1),Y	an augenblickliche Position schreiben
E81F	AD 86 02	LDA \$0286	Farbe
E822	91 F3	STA (\$F3),Y	setzen
E824	E6 D8	INC \$D8	Anzahl der Inserts erhöhen
E826	4C A8 E6	JMP \$E6A8	
E829	A6 D8	LDX \$D8	
E82B	F0 05	BEQ \$E832	
E82D	09 40	ORA #\$40	
E82F	4C 97 E6	JMP \$E697	
*****			
E832	C9 11	CMP \$11	Cursor up
E834	D0 16	BNE \$E84C	
E836	A6 D6	LDX \$D6	Zeile
E838	F0 37	BEQ \$E871	null, dann fertig
E83A	C6 D6	DEC \$D6	Zeilennummer eins erniedrigen
E83C	A5 D3	LDA \$D3	Spalte
E83E	38	SEC	
E83F	E9 28	SBC #\$28	40 abziehen
E841	90 04	BCC \$E847	

E843	B5 D3	STA #D3	Cursorspalte
E845	10 2A	BPL #E871	positiv, ok
E847	20 6C E5	JSR #E56C	Bildschirmzeiger neu setzen
E84A	D0 25	BNE #E871	
E84C	C9 12	CMP #12	'RVS OFF'
E84E	D0 04	BNE #E854	
E850	A9 00	LDA #00	
E852	B5 C7	STA #C7	RVS-Flag löschen
E854	C9 1D	CMP #1D	'Cursor left' ?
E856	D0 12	BNE #E86A	
E858	98	TYA	
E859	F0 09	BEQ #E864	
E85E	20 A1 E8	JSR #E8A1	
E85E	88	DEY	
E85F	B4 D3	STY #D3	Cursorspalte
E861	4C A8 E6	JMP #E6A8	fertig
E864	20 01 E7	JSR #E701	Ruckschrit in vorherige Zeile
E867	4C A8 E6	JMP #E6A8	fertig
E86A	C9 13	CMP #13	'CLR SCREEN' ?
E86C	D0 06	BNE #E874	
E86E	20 44 E5	JSR #E544	Bildschirm löschen
E871	4C A8 E6	JMP #E6A8	fertig
E874	09 80	ORA #80	Bit 7 wieder herstellen
E876	20 CB EB	JSR #E8CB	auf Farbcode prüfen
E879	4C 4F EC	JMP #EC4F	prüft auf Umschaltung Text/Grafik
E87C	46 C9	LSR #C9	
E87E	A6 D6	LDX #D6	
E880	EB	INX	
E881	E0 19	CPX #19	25, letzte Zeile
E883	D0 03	BNE #E88B	
E885	20 EA E8	JSR #E8EA	Bildschirm scrollen
E88E	B5 D9	LDA #D9,X	
E88A	10 F4	BPL #E880	
E88C	B6 D6	STX #D6	
E88E	4C 6C E5	JMP #E56C	Cursorposition berechnen
E891	A2 00	LDX #00	
E893	B6 D8	STX #D8	
E895	B6 C7	STX #C7	Flags löschen
E897	B6 D4	STX #D4	
E899	B6 D3	STX #D3	
E89B	20 7C EB	JSR #E87C	
E89E	4C A8 E6	JMP #E6A8	fertig
E8A1	A2 02	LDX #02	
E8A3	A9 00	LDA #00	
E8A5	C5 D3	CMP #D3	
E8A7	F0 07	BEQ #E8B0	
E8A9	18	CLC	
E8AA	69 28	ADC #28	40 addieren, eine Zeile
E8AC	CA	DEX	
E8AD	D0 F6	BNE #E8A5	
E8AF	60	RTS	
E8B0	C6 D6	DEC #D6	
E8B2	60	RTS	
E8B3	A2 02	LDX #02	
E8B5	A9 27	LDA #27	39, letzte Spalte
E8B7	C5 D3	CMP #D3	
E8B9	F0 07	BEQ #E8C2	
E8BB	18	CLC	
E8BC	69 28	ADC #28	40 addieren
E8BE	CA	DEX	

E8BF	D0 F6	BNE \$E8B7	
E8C1	60	RTS	
E8C2	46 D6	LDX \$D6	
E8C4	E0 19	CPX \$19	25
E8C6	F0 02	BEQ \$E8CA	
E8C8	E6 D6	INC \$D6	
E8CA	60	RTS	
***** prüft auf Farbcodes			
E8CB	A2 0F	LDX \$0F	Anzahl der Codes
E8CD	DD DA E8	CMF \$E8DA,X	mit Farbkodetabelle vergleichen
E8D0	F0 04	BEQ \$E8D6	gefunden
E8D2	CA	DEX	
E8D3	10 FB	BPL \$E8CD	
E8D5	60	RTS	
E8D6	BE 86 02	STX \$0286	Farb-Code setzen
E8D9	60	RTS	
***** Tabelle der Farb-Codes			
E8DA	90 05 1C 9F 9C 1E 1F 9E		
E8E2	81 95 96 97 98 99 9A 9B		
***** Bildschirm Scrollen			
E8EA	A5 AC	LDA \$AC	
E8EC	48	PHA	
E8ED	A5 AD	LDA \$AD	
E8EF	48	PHA	Zeiger retten
E8F0	A5 AE	LDA \$AE	
E8F2	48	PHA	
E8F3	A5 AF	LDA \$AF	
E8F5	48	PHA	
E8F6	A2 FF	LDX \$FF	ab Zeile Null beginnen
E8F8	C6 D6	DEC \$D6	Zeilennummer erniedrigen
E8FA	C6 C9	DEC \$C9	
E8FC	CE A5 02	DEC \$02A5	
E8FF	E8	INX	Zeilennummer erhöhen
E900	20 F0 E9	JSR \$E9F0	Zeiger auf Videoram für Zeile X
E903	E0 18	CPX \$18	24
E905	B0 0C	BCS \$E913	schon alle Zeilen ?
E907	BD F1 EC	LDA \$ECF1,X	LSB holen
E90A	B5 AC	STA \$AC	
E90C	B5 DA	LDA \$DA,X	MSB
E90E	20 C8 E9	JSR \$E9C8	Bildschirmzeile nach oben schieben
E911	30 EC	BM1 \$E8FF	nachste Zeile
E913	20 FF E9	JSR \$E9FF	unterste Bildschirmzeile löschen
E916	A2 00	LDX \$00	
E918	B5 D9	LDA \$D9,X	
E91A	29 7F	AND \$7F	
E91C	B4 DA	LDY \$DA,X	
E91E	10 02	BPL \$E922	
E920	09 80	ORA \$80	
E922	95 D9	STA \$D9,X	
E924	E8	INX	
E925	E0 18	CPX \$18	24
E927	D0 EF	BNE \$E918	
E929	A5 F1	LDA \$F1	
E92B	09 80	ORA \$80	
E92D	B5 F1	STA \$F1	
E92F	A5 D9	LDA \$D9	
E931	10 C3	BPL \$E8F6	

E933	E6 D6	INC \$D6	
E935	EE A5 02	INC \$02A5	
E938	A9 7F	LDA #\$7F	
E93A	8D 00 DC	STA \$DC00	
E93D	AD 01 DC	LDA \$DC01	
E940	C9 FB	CMP #\$FB	CTRL-Taste gedruckt ?
E942	08	PHF	
E943	A9 7F	LDA #\$7F	
E945	8D 00 DC	STA \$DC00	
E948	28	PLP	
E949	D0 08	BNE \$E956	nicht gedruckt
E94B	A0 00	LDY #\$00	
E94D	EA	NOP	
E94E	CA	DEX	
E94F	D0 FC	BNE \$E94D	Verzögerungsschleife
E951	88	DEY	
E952	D0 F9	BNE \$E94D	
E954	84 C6	STY \$C6	Anzahl der gedruckten Tasten gleich null
E956	A6 D6	LDX \$D6	
E958	68	PLA	
E959	85 AF	STA \$AF	
E95B	68	PLA	
E95C	85 AE	STA \$AE	Zeiger zurückholen
E95E	68	PLA	
E95F	85 AD	STA \$AD	
E961	68	PLA	
E962	85 AC	STA \$AC	
E964	60	RTS	

\*\*\*\*\* Einfügen einer Fortsetzungszeile  
Zeilennummer

E965	A6 D6	LDX \$D6	
E967	E8	INX	
E968	B5 D9	LDA \$D9,X	
E96A	10 FB	BPL \$E967	
E96C	BE A5 02	STX \$02A5	
E96F	E0 18	CPX #\$18	24
E971	F0 0E	BEQ \$E981	
E973	90 0C	BCC \$E981	
E975	20 EA E8	JSR \$E8EA	Bildschirm Scrollen
E978	AE A5 02	LDX \$02A5	
E97B	CA	DEX	
E97C	C6 D6	DEC \$D6	Zeilennummer erniedrigen
E97E	4C DA E6	JMP \$E6DA	

\*\*\*\*\*

E981	A5 AC	LDA \$AC	
E983	48	PHA	
E984	A5 AD	LDA \$AD	
E986	48	PHA	
E987	A5 AE	LDA \$AE	
E989	48	PHA	
E98A	A5 AF	LDA \$AF	
E98C	48	PHA	
E98D	A2 19	LDX #\$19	25
E98F	CA	DEX	Zeilennummer
E990	20 F0 E9	JSR \$E9F0	Farbraum Zeiger berechnen
E993	EC A5 02	CPX \$02A5	
E996	90 0E	BCC \$E9A6	
E998	F0 0C	BEQ \$E9A6	
E99A	8D EF EC	LDA \$ECEF,X	LSB des Zeilenanfangs setzen

E99D	85 AC	STA \$AC	
E99F	85 D8	LDA \$D8,Y	MSB setzen
E9A1	20 C8 E9	JSR \$E9C8	Zeile nach oben schieben
E9A4	30 E9	BNL \$E9BF	
E9A6	20 FF E9	JSR \$E9FF	Bildschirmzeile löschen
E9A9	A2 17	LDX #17	23
E9AB	EC A5 02	CPX \$02A5	
E9AE	90 0F	BCC \$E9BF	
E9B0	85 DA	LDA \$DA,X	
E9B2	29 7F	AND #7F	
E9B4	B4 D9	LDY \$D9,X	
E9B6	10 02	BPL \$E9BA	
E9B8	09 80	ORA #80	
E9BA	95 DA	STA \$DA,X	
E9BC	CA	DEX	
E9BD	D0 EC	BNE \$E9AD	
E9BF	AE A5 02	LDX \$02A5	
E9C2	20 DA E6	JSR \$E6DA	MSB neu berechnen
E9C5	4C 58 E9	JMP \$E958	Register zurückholen, RTS

*****			Zeile nach oben schieben
E9C8	29 03	AND #03	
E9CA	0D 88 02	ORA \$0288	Bildschirmzeiger für neue Zeile
E9CD	85 AD	STA \$AD	
E9CF	20 E0 E9	JSR \$E9E0	Zeiger für neue Zeile berechnen
E9D2	A0 27	LDY #27	39 Zeichen
E9D4	B1 AC	LDA (\$AC),Y	
E9D6	91 D1	STA (\$D1),Y	Zeichen
E9D8	B1 AE	LDA (\$AE),Y	
E9DA	91 F3	STA (\$F3),Y	und Farbe übertragen
E9DC	88	DEY	
E9DD	10 F3	BPL \$E9D4	alle Spalten ?
E9DF	60	RTS	

*****			Bildschirmzeile für Scrollzeile berechnen
E9E0	20 24 EA	JSR \$EA24	Zeiger auf Farbram berechnen
E9E3	A5 AC	LDA \$AC	
E9E5	85 AE	STA \$AE	
E9E7	A5 AD	LDA \$AD	
E9E9	29 03	AND #03	Zeiger in \$AE/\$AF
E9EB	09 D8	ORA #D8	
E9ED	85 AF	STA \$AF	
E9EF	60	RTS	

*****			Zeiger auf Videoram für Zeile X
E9F0	BD F0 EC	LDA \$ECF0,X	LSB holen
E9F3	85 D1	STA \$D1	
E9F5	B5 D9	LDA \$D9,X	MSB
E9F7	29 03	AND #03	
E9F9	0D 88 02	ORA \$0288	Highbyte des Videorams
E9FC	85 D2	STA \$D2	
E9FE	60	RTS	

*****			Bildschirmzeile X löschen
E9FF	A0 27	LDY #27	39 Spalten
EA01	20 F0 E9	JSR \$E9F0	Zeiger auf Videoram setzen
EA04	20 24 EA	JSR \$EA24	Zeiger auf Farbram setzen
EA07	A9 20	LDA #20	Leerzeichen
EA09	91 D1	STA (\$D1),Y	setzen
EA0B	20 DA E4	JSR \$E4DA	Hintergrundfarbe setzen

EA0E	EA	NOP	
EA0F	88	DEY	
EA10	10 F5	BPL \$EA07	schon Spalte 0 ?
EA12	60	RTS	
*****			
EA13	A8	TAY	
EA14	A9 02	LDA #02	
EA16	85 CD	STA \$CD	Blinkzähler bei Repeatfunktion setzen
EA18	20 24 EA	JSR \$EA24	Zeiger auf Farbram berechnen
EA1B	98	TYA	
*****			
EA1C	A4 D3	LDY \$D3	Zeichen und Farbe auf Bildschirm setzen
EA1E	91 D1	STA (\$D1),Y	Spaltenposition
EA20	8A	TXA	Zeichen in Akku auf Bildschirm
EA21	91 F3	STA (\$F3),Y	Farb-Kode in X
EA23	60	RTS	in Farb-RAM schreiben
*****			
EA24	A5 D1	LDA \$D1	Zeiger auf Farb-RAM berechnen
EA26	85 F3	STA \$F3	\$D1/\$D2 = Zeiger auf Video-RAM-Position
EA28	A5 D2	LDA \$D2	
EA2A	29 03	AND #03	
EA2C	09 DB	ORA #0DB	High-Byte = \$DB
EA2E	85 F4	STA \$F4	\$F3/\$F4 = Zeiger auf Farb-RAM-Position
EA30	60	RTS	
*****			
EA31	20 EA FF	JSR \$FFEA	Interrupt-Routine
EA34	A5 CC	LDA \$CC	Stop-Taste, Zeit erhöhen
EA36	D0 29	BNE \$EA61	Blink-Flag für Cursor
EA38	C6 CD	DEC \$CD	nicht blinkend, dann weiter
EA3A	D0 25	BNE \$EA61	Blinkzähler erniedrigen
EA3C	A9 14	LDA #14	nicht null, dann weiter
EA3E	85 CD	STA \$CD	Blinkzähler wieder auf 20 setzen
EA40	A4 D3	LDY \$D3	Cursorspalte
EA42	46 CF	LSR \$CF	Blinkschalter null dann C=1
EA44	AE 87 02	LDX \$0287	Farbe unter Cursor
EA47	B1 D1	LDA (\$D1),Y	Zeichen-Kode setzen
EA49	B0 11	BCS \$EA5C	Blinkschalter war ein, dann weiter
EA4B	E6 CF	INC \$CF	Blinkschalter ein
EA4D	85 CE	STA \$CE	Zeichen unter Cursor merken
EA4F	20 24 EA	JSR \$EA24	Zeiger in Farb-RAM berechnen
EA52	B1 F3	LDA (\$F3),Y	Farb-Kode holen
EA54	8D 87 02	STA \$0287	und merken
EA57	AE 86 02	LDX \$0286	Farb-Kode unter Cursor
EA5A	A5 CE	LDA \$CE	Zeichen unter Cursor
EA5C	49 80	EOR #80	RVS-Bit umdrehen
EA5E	20 1C EA	JSR \$EA1C	Zeichen und Farbe setzen
EA61	A5 01	LDA \$01	
EA63	29 10	AND #10	prüft Rekorder-Taste
EA65	F0 0A	BEQ \$EA71	gedruckt ?
EA67	A0 00	LDY #00	
EA69	84 C0	STY \$C0	Rekorder-Flag setzen
EA6B	A5 01	LDA \$01	
EA6D	09 20	ORA #20	Rekoder-Motor aus
EA6F	D0 08	BNE \$EA79	
EA71	A5 C0	LDA \$C0	Rekorder-Flag
EA73	D0 06	BNE \$EA7B	
EA75	A5 01	LDA \$01	



EA77	29 1F	AND #1F	Rekorder-Motor ein
EA79	B5 01	STA #01	
EA7B	20 B7 EA	JSR \$EAB7	Tastatur-Abfrage
EA7E	AD 0D DC	LDA \$DC0D	IRQ-Flag löschen
EAB1	6B	PLA	
EAB2	A8	TAY	
EAB3	6B	PLA	Register wieder herstellen
EAB4	AA	TAX	
EAB5	6B	PLA	und Rückkehr vom Interrupt
EAB6	40	RTI	

\*\*\*\*\*

Tastatur-Abfrage

EAB7	A9 00	LDA #00	
EAB9	BD BD 02	STA \$02BD	Shift/CTRL Flag rucksetzen
EABC	A0 40	LDY #40	\$40 = keine Taste gedrückt
EABE	B4 CB	STY \$CB	Kode für gedrückte Taste
EAC0	BD 00 DC	STA \$DC00	jede Matrixzeile testen
EAC3	AE 01 DC	LDX \$DC01	
EAC6	E0 FF	CPX #FF	keine Taste gedrückt ?
EAC8	F0 61	BEQ \$EAFB	dann beenden
EACA	A8	TAY	
EACB	A9 B1	LDA #B1	
EACD	B5 F5	STA \$F5	
EAEF	A9 EB	LDA #EB	
EAA1	B5 F6	STA \$F6	Zeiger auf Tabelle 1 \$EB81
EAA3	A9 FE	LDA #FE	
EAA5	BD 00 DC	STA \$DC00	
EAA8	A2 08	LDX #08	8 Matrixzeilen
EAAA	4B	PHA	
EACB	AD 01 DC	LDA \$DC01	
EAAE	CD 01 DC	CMP \$DC01	Tastatur entprellen
EAB1	D0 F8	BNE \$EAB8	
EAB3	4A	LSR A	Bits nacheinander ins Carry schieben
EAB4	B0 16	BCS \$EACC	'1' gleich nicht gedrückt
EAB6	4B	PHA	
EAB7	B1 F5	LDA (\$F5),Y	ASCII-Kode aus Tabelle holen
EAB9	C9 05	CMP #05	
EABB	B0 0C	BCS \$EAC9	großer gleich 5 ?
EABD	C9 03	CMP #03	
EABF	F0 0B	BEQ \$EAC9	'STOP'-Kode ?
EAC1	0D BD 02	ORA \$02BD	
EAC4	BD BD 02	STA \$02BD	Flag setzen
EAC7	10 02	BPL \$EACB	
EAC9	B4 CB	STY \$CB	Nummer der Taste merken
EACB	6B	PLA	
EACC	CB	INY	
EACD	C0 41	CPY #41	
EACF	B0 0B	BCS \$EADC	großer \$40 ?
EAD1	CA	DEX	
EAD2	D0 DF	BNE \$EAB3	nächste Matrix-Spalte
EAD4	3B	SEC	
EAD5	6B	PLA	
EAD6	2A	ROL A	
EAD7	BD 00 DC	STA \$DC00	
EADA	D0 CC	BNE \$EAB8	nächste Matrix-Zeile
EADC	6B	PLA	
EADD	6C BF 02	JMP (\$02BF)	JMP \$EB4B setzt Zeiger auf Tabelle
EAE0	A4 CB	LDY \$CB	Nummer der Taste
EAE2	B1 F5	LDA (\$F5),Y	ASCII-Wert aus Tabelle holen
EAE4	AA	TAX	

EAE5	C4 C5	CPY \$C5	mit letzter Taste vergleichen
EAE7	F0 07	BEQ \$EAF0	
EAE9	A0 10	LDY \$010	
EAE8	8C 8C 02	STY \$028C	Repeat-Verzögerungszähler
EAE6	D0 36	BNE \$EB26	
EAF0	29 7F	AND \$07F	Bit 7 löschen
EAF2	2C 8A 02	BIT \$028A	Repeat-Funktion für alle Tasten ?
EAF5	30 16	BMI \$EB0D	Bit 7 gesetzt, dann alle Taste wiederholen
EAF7	70 49	BVS \$EB42	Bit 6 gesetzt, dann ignorieren
EAF9	C9 7F	CMP \$07F	nur folgende Tasten wiederholen
EAFB	F0 29	BEQ \$EB26	
EAFD	C9 14	CMP \$014	'DEL', 'INST' Kode
EAF7	F0 0C	BEQ \$EB0D	
EB01	C9 20	CMP \$020	Leerzeichen
EB03	F0 08	BEQ \$EB0D	
EB05	C9 1D	CMP \$01D	Cursor right, left
EB07	F0 04	BEQ \$EB0D	
EB09	C9 11	CMP \$011	Cursor down, up
EB0B	D0 35	BNE \$EB42	
EB0D	AC 8C 02	LDY \$028C	Repeatverzögerungszähler
EB10	F0 05	BEQ \$EB17	
EB12	CE 8C 02	DEC \$028C	runterzahlen
EB15	D0 2B	BNE \$EB42	
EB17	CE 8B 02	DEC \$028B	Repeatgeschwindigkeitsszähler
EB1A	D0 26	BNE \$EB42	
EB1C	A0 04	LDY \$004	
EB1E	BC 8B 02	STY \$028B	Zähler neu setzen
EB21	A4 C6	LDY \$C6	Anzahl der Zeichen im Tastaturpuffer
EB23	8B	DEY	
EB24	10 1C	BPL \$EB42	mehr als ein Zeichen im Puffer, dann ignorieren
EB26	A4 C6	LDY \$C6	
EB28	B4 C5	STY \$C5	
EB2A	AC 8D 02	LDY \$028D	
EB2D	8C BE 02	STY \$028E	
EB30	E0 FF	CPX \$0FF	Tastatur-Kode ungültig ?
EB32	F0 0E	BEQ \$EB42	ja, dann ignorieren
EB34	8A	TXA	
EB35	A6 C6	LXI \$C6	Anzahl der Zeichen im Tastaturpuffer
EB37	EC 89 02	CPX \$0289	mit Maximalzahl vergleichen
EB3A	B0 06	BCS \$EB42	Puffer voll, dann Zeichen ignorieren
EB3C	9D 77 02	STA \$0277,X	Zeichen in Tastaturpuffer schreiben
EB3F	EB	INX	
EB40	B6 C6	STX \$C6	Zeichenzahl erhöhen
EB42	A9 7F	LDA \$07F	Tastatur-Matrix Abfrage auf Default
EB44	8D 00 DC	STA \$DC00	
EB47	60	RTS	

*****			Prüft auf Shift, CTRL, Commodore
EB48	AD 8D 02	LDA \$028D	Flag für Shift/CTRL
EB4B	C9 03	CMP \$003	
EB4D	D0 15	BNE \$EB64	Zeiger auf Dekodiertabelle berechnen
EB4F	CD BE 02	CMP \$02BE	
EB52	F0 EE	BEQ \$EB42	
EB54	AD 91 02	LDA \$0291	Shift-Commodore erlaubt ?
EB57	30 1D	BMI \$EB76	nein, zurück zur Dekodierung
EB59	AD 18 D0	LDA \$D018	Shift/Commodore
EB5C	49 02	EOR \$002	Umschaltung Klein/Groß -Schreibung
EB5E	8D 18 D0	STA \$D018	
EB61	4C 76 EB	JMP \$EB76	fertig
EB64	0A	ASL A	

```

EB65 C9 00      CMP #00
EB67 90 02      BCC $EB6B
EB69 A9 06      LDA #06
EB6B AA        TAX
EB6C B0 79 EB   LDA $EB79,X
EB6F B5 F5      STA $F5      Zeiger auf Tastatur-Dekodiertabelle laden
EB71 B0 7A EB   LDA $EB7A,X
EB74 B5 F6      STA $F6
EB76 4C E0 EA   JMP $EAE0    zuruck zur Dekodierung

```

```

***** Zeiger auf Tastatur-Dekodiertabellen
EB79 B1 EB C2 EB 03 EC 7B EC

```

```

***** Tastatur-Dekodiertabelle 1, ungeshifted
EB81 14 0D 1D 08 B5 B6 B7 11
EB89 33 57 41 34 5A 53 45 01
EB91 35 52 44 36 43 46 54 5B
EB99 37 59 47 38 42 4B 55 56
EBA1 39 49 4A 30 4D 4B 4F 4E
EBA9 2B 50 4C 2D 2E 3A 40 2C
EBB1 5C 2A 3B 13 01 3D 5E 2F
EBB9 31 5F 04 32 20 02 51 03
EBC1 FF

```

```

***** Tastatur-Dekodierung, Tabelle 2 geshifted
EBC2 94 BD 9D BC B9 BA BB 91
EBC9 23 D7 C1 24 DA D3 C5 01
EBD2 25 D2 C4 26 C3 C6 D4 DB
EBDA 27 D9 C7 28 C2 C8 D5 D6
EBE2 29 C9 CA 30 CD CB CF CE
EBEA DB D0 CC DD 3E 5B BA 3C
EBF2 A9 C0 5D 93 01 3D DE 3F
EBFA 21 5F 04 22 A0 02 D1 B3
EC02 FF

```

```

***** Tastatur-Dekodierung, Tabelle 3, mit 'C'-Taste
EC03 94 BD 9D BC B9 BA BB 91
EC0B 96 B3 B0 97 AD AE B1 01
EC13 9B B2 AC 99 BC BB A3 BD
EC1B 9A B7 A5 9B BF B4 BB BE
EC23 29 A2 B5 30 A7 A1 B9 AA
EC2B A6 AF B6 DC 3E 5B AA 3C
EC33 AB DF 5D 93 01 3D DE 3F
EC3B B1 5F 04 95 A0 02 AB B3
EC43 FF

```

```

***** prüft auf Steuerzeichen
EC44 C9 0E      CMP #0E      chr$(14)
EC46 D0 07      BNE $EC4F
EC48 AD 1B D0   LDA $D01B    Character-Generator
EC4B 09 02      ORA #02      auf Großschrift-Modus
EC4D D0 09      BNE $EC5B
EC4F C9 0E      CMP #0E      chr$(142)
EC51 D0 0B      BNE $EC5E
EC53 AD 1B D0   LDA $D01B
EC56 29 FD      AND #FD      Kleinschrift-Modus
EC5B BD 1B D0   STA $D01B    setzen
EC5B 4C AB E6   JMP $E6AB
EC5E C9 08      CMP #08      chr$(8)
EC60 D0 07      BNE $EC69

```

EC62	A9 80	LDA #\$80	
EC64	0D 91 02	ORA \$0291	Shift-Commodore sperren
EC67	30 09	BMI \$EC72	
EC69	C9 09	CMP #\$09	chr\$(9)
EC6B	D0 EE	BNE \$ECSB	
EC6D	A9 7F	LDA #\$7F	
EC6F	2D 91 02	AND \$0291	Shift-Commodore ermöglichen
EC72	BD 91 02	STA \$0291	
EC75	4C AB E6	JMP \$E6AB	

*****			Tastaturdekodierung, Tabelle 4, mit CTRL-Taste
EC7B	FF FF FF FF FF FF FF FF		
EC80	1C 17 01 9F 1A 13 05 FF		
EC8B	9C 12 04 1E 03 06 14 1B		
EC90	1F 19 07 9E 02 1B 15 16		
EC98	12 09 0A 92 0D 0B 0F 0E		
ECA0	FF 10 0C FF FF 1B 00 FF		
ECA8	1C FF 1D FF FF 1F 1E FF		
ECB0	90 06 FF 05 FF FF 11 FF		
ECB8	FF		

*****			Konstanten für Videocontroller
ECB9	00 00 00 00 00 00 00 00		
ECC1	00 00 00 00 00 00 00 00		
ECC9	00 9B 37 00 00 00 0B 00		
ECD1	14 0F 00 00 00 00 00 00		
ECD9	0E 06 01 02 03 04 00 01		
ECE1	02 03 04 05 06 07		

*****			Text nach Drucken von SHIFT RUN/STOP
ECE7	4C 4F 41 44 0D 52 55 4E		'load (cr) run (cr)'
ECEA	0D		

*****			Tabelle der LSB der Bildschirmzeilen-Anfänge
ECF0	00 2B 50 7B A0 C8 F0 1B		
ECF8	40 6B 90 BB E0 0B 30 5B		
ED00	B0 AB D0 F8 20 4B 70 9B		
ED0B	C0		

*****			IEC-Bus Routinen
-------	--	--	------------------

*****			TALK senden
ED09	09 40	GRA #\$40	Bit für Talk setzen
ED0B	2C	,BYTE \$2C	

*****			LISTEN senden
ED0C	09 20	ORA #\$20	Bit für Listen setzen
ED0E	20 A4 F0	JSR \$F0A4	Timer für IEC Time-out setzen
ED11	4B	PHA	
ED12	24 94	BIT \$94	noch ein Byte auszugeben ?
ED14	10 0A	BPL \$ED20	nein
ED16	3B	SEC	
ED17	66 A3	ROR \$A3	
ED19	20 40 ED	JSR \$ED40	Byte auf IEC-Bus ausgeben
ED1C	46 94	LSR \$94	
ED1E	46 A3	LSR \$A3	
ED20	6B	PLA	
ED21	85 95	STA \$95	auszugebendes Byte
ED23	7B	SEI	
ED24	20 97 EE	JSR \$EE97	

```

ED27 C9 3F      CMP #3F
ED29 D0 03      BNE $ED2E
ED2B 20 B5 EE    JSR $EEB5
ED2E AD 00 DD    LDA $DD00
ED31 09 0B      ORA #0B
ED33 8D 00 DD    STA $DD00
ED36 7B         SEI
ED37 20 BE EE    JSR $EEBE
ED3A 20 97 EE    JSR $EE97
ED3D 20 B3 EE    JSR $EEB3

```

ATN setzen

\*\*\*\*\*

ein Byte auf IEC-Bus ausgeben

```

ED40 7B         SEI
ED41 20 97 EE    JSR $EE97
ED44 20 A9 EE    JSR $EEA9
ED47 B0 64      BCS $EDAD
ED49 20 B5 EE    JSR $EEB5
ED4C 24 A3      BIT #A3
ED4E 10 0A      BPL $ED5A
ED50 20 A9 EE    JSR $EEA9
ED52 90 FB      BCC $ED50
ED55 20 A9 EE    JSR $EEA9
ED58 B0 FB      BCS $ED55
ED5A 20 A9 EE    JSR $EEA9
ED5D 90 FB      BCC $ED5A
ED5F 20 BE EE    JSR $EEBE
ED62 A9 0B      LDA #0B
ED64 B5 A5      STA #A5
ED66 AD 00 DD    LDA $DD00
ED69 CD 00 DD    CMP $DD00
ED6C D0 FB      BNE $ED66
ED6E 0A         ASL A
ED6F 90 3F      BCC $EDB0
ED71 66 95      ROR #95
ED73 B0 05      BCS $ED7A
ED75 20 A0 EE    JSR $EEA0
ED7B D0 03      BNE $ED7D
ED7A 20 97 EE    JSR $EE97
ED7D 20 B5 EE    JSR $EEB5
ED80 EA         NOP
ED81 EA         NOP
ED82 EA         NOP
ED83 EA         NOP
ED84 AD 00 DD    LDA $DD00
ED87 29 DF      AND #DF
ED89 09 10      ORA #10
ED8B BD 00 DD    STA $DD00
ED8E C6 A5      DEC #A5
ED90 D0 D4      BNE $ED66
ED92 A9 04      LDA #04
ED94 BD 07 DC    STA $DC07
ED97 A9 19      LDA #19
ED99 BD 0F DC    STA $DC0F
ED9C AD 0D DC    LDA $DC0D
ED9F AD 0D DC    LDA $DC0D
EDA2 29 02      AND #02
EDA4 D0 0A      BNE $EDB0
EDA6 20 A9 EE    JSR $EEA9
EDA9 B0 F4      BCS $ED9F
EDAB 5B         CLI
EDAC 60         RTS

```

Bitzähler für serielle Ausgabe setzen

Daten ausgeben. '1'  
Clock aus

\*\*\*\*\*

ED4D	A9 80	LDA #80	'device not present'
EDAF	2C	.BYTE \$2C	
EDB0	A9 03	LDA #03	'time out'
EDB2	20 1C FE	JSR \$FE1C	Status setzen
EDB5	58	CLI	
EDB6	18	CLC	
EDB7	90 4A	BCC \$EE03	

\*\*\*\*\*

EDB9	85 95	STA \$95	Sekundaradresse nach LISTEN senden
EDBB	20 36 ED	JSR \$ED36	Sekundaradresse speichern
EDBE	AD 00 DD	LDA \$DD00	mit ATN ausgeben
EDC1	29 F7	AND #F7	ATN rucksetzen
EDC3	BD 00 DD	STA \$DD00	
EDC6	60	RTS	

\*\*\*\*\*

EDC7	85 95	STA \$95	Sekundaradresse nach TALK ausgeben
EDC9	20 36 ED	JSR \$ED36	Sekundaradresse speichern
EDCC	78	SEI	mit ATN ausgeben
EDCD	20 A0 EE	JSR \$EEA0	Bit '0' ausgeben
EDD0	20 BE ED	JSR \$EDBE	ATN rucksetzen
EDD3	20 85 EE	JSR \$EE85	Takt ein
EDD6	20 A9 EE	JSR \$EEA9	Datenbit holen
EDD9	30 FB	BMI \$EDD6	ja ?
EDDB	58	CLI	
EDDC	60	RTS	

\*\*\*\*\*

EDDD	24 94	BIT \$94	IECOUT ein Byte auf IEC-Bus ausgeben
EDDF	30 05	BMI \$EDE6	
EDE1	38	SEC	
EDE2	66 94	ROR \$94	
EDE4	D0 05	BNE \$EDEB	
EDE6	48	PHA	
EDE7	20 40 ED	JSR \$ED40	Byte auf Bus geben
EDEA	68	PLA	
EDEB	85 95	STA \$95	
EDED	18	CLC	
EDEE	60	RTS	

\*\*\*\*\*

EDEF	78	SEI	UNTALK senden
EDF0	20 BE EE	JSR \$EEBE	Takt aus
EDF3	AD 00 DD	LDA \$DD00	
EDF6	09 08	DRA #08	ATN setzen
EDF8	BD 00 DD	STA \$DD00	
EDFB	A9 5F	LDA #5F	
EDFD	2C	.BYTE \$2C	

\*\*\*\*\*

EDFE	A9 3F	LDA #3F	UNLISTEN senden
EE00	20 11 ED	JSR \$ED11	ausgeben
EE03	20 BE ED	JSR \$EDBE	ATN rucksetzen
EE06	8A	TXA	
EE07	A2 0A	LDX #0A	
EE09	CA	DEX	ca. 40 Mikrosekunden warten
EE0A	D0 FD	BNE \$EE09	
EE0C	AA	TAX	

EE0D	20 85 EE	JSR \$EE85	Takt ein
EE10	4C 97 EE	JMP \$EE97	
*****			IECIN ein Zeichen vom IEC-Bus holen
EE13	78	SEI	
EE14	A9 00	LDA #00	
EE16	85 A5	STA \$A5	
EE18	20 85 EE	JSR \$EE85	Takt ausgeben
EE18	20 A9 EE	JSR \$EEA9	Datenbit holen
EE1E	10 FB	BPL \$EE1B	
EE20	A9 01	LDA #01	
EE22	8D 07 DC	STA \$DC07	
EE25	A9 19	LDA #19	
EE27	8D 0F DC	STA \$DC0F	
EE2A	20 97 EE	JSR \$EE97	Bit '1' ausgeben
EE2D	AD 0D DC	LDA \$DC0D	
EE30	AD 0D DC	LDA \$DC0D	Timer abfragen
EE33	29 02	AND #02	
EE35	D0 07	BNE \$EE3E	
EE37	20 A9 EE	JSR \$EEA9	Datenbit holen
EE3A	30 F4	BMI \$EE30	
EE3C	10 18	BPL \$EE56	
EE3E	A5 A5	LDA \$A5	
EE40	F0 05	BEQ \$EE47	
EE42	A9 02	LDA #02	'time out'
EE44	4C B2 ED	JMP \$EDB2	Status setzen
EE47	20 A0 EE	JSR \$EEA0	Bit '0' ausgeben
EE4A	20 85 EE	JSR \$EE85	Takt ein
EE4D	A9 40	LDA #40	'EOF'
EE4F	20 1C FE	JSR \$FE1C	Status setzen
EE52	E6 A5	INC \$A5	
EE54	D0 CA	BNE \$EE20	
EE56	A9 08	LDA #08	
EE58	B5 A5	STA \$A5	Bitzähler setzen
EE5A	AD 00 DD	LDA \$DD00	Datenbit holen
EE5D	CD 00 DD	CHP \$DD00	
EE60	D0 F8	BNE \$EE5A	
EE62	0A	ASL A	ins Carry schieben
EE63	10 F5	BPL \$EE5A	
EE65	66 A4	ROR \$A4	
EE67	AD 00 DD	LDA \$DD00	
EE6A	CD 00 DD	CHP \$DD00	Daten holen und entprellen
EE6D	D0 F8	BNE \$EE67	
EE6F	0A	ASL A	
EE70	30 F5	BMI \$EE67	
EE72	C6 A5	DEC \$A5	
EE74	D0 E4	BNE \$EE5A	
EE76	20 A0 EE	JSR \$EEA0	
EE79	24 90	BIT #90	Status
EE7B	50 03	BVC \$EE80	kein EOF ?
EE7D	20 06 EE	JSR \$EE06	warten und Bit '1' senden
EE80	A5 A4	LDA \$A4	
EE82	58	CLI	
EE83	18	CLC	
EE84	60	RTS	
*****			Seriellen Takt ein
EE85	AD 00 DD	LDA \$DD00	
EE88	29 EF	AND #EF	Bit 4 löschen
EE8A	8D 00 DD	STA \$DD00	

EE8D	60	RTS	
*****			Seriellen Takt aus
EE8E	AD 00 DD	LDA #DD00	
EE91	09 10	ORA #10	Bit 4 setzen
EE93	BD 00 DD	STA #DD00	
EE96	60	RTS	
*****			'1'-Bit ausgeben
EE97	AD 00 DD	LDA #DD00	
EE9A	29 DF	AND #DF	Bit 5 löschen
EE9C	BD 00 DD	STA #DD00	
EE9F	60	RTS	
*****			'0'-Bit ausgeben
EEA0	AD 00 DD	LDA #DD00	
EEA3	09 20	ORA #20	Bit 5 setzen
EEA5	BD 00 DD	STA #DD00	
EEA8	60	RTS	
*****			Bit von IEC-Bus ins Carry-Flag holen
EEA9	AD 00 DD	LDA #DE00	
EEAC	CD 00 DD	CMP #DD00	Datenregister
EEAF	D0 FB	BNE \$EEA9	
EEB1	0A	ASL A	Datenbit ins Carry schieben
EEB2	60	RTS	
*****			Verzögerung 1 Millisekunde
EEB3	BA	TXA	
EEB4	A2 BB	LXI #BBB	104
EEB6	CA	DEX	
EEB7	D0 FD	BNE \$EEB6	
EEB9	AA	TAY	
EEBA	60	RTS	
*****			RS 232 Ausgabe
EEB8	A5 B4	LDA #B4	
EEBD	F0 47	BEQ \$EF06	
EEBF	30 3F	BMI \$EF00	
EEC1	46 B6	LSR #B6	
EEC3	A2 00	LXI #00	
EEC5	90 01	BCC \$EECB	
EEC7	CA	DEX	
EECB	BA	TXA	
EEC9	45 BD	EOR #BD	
EECB	85 9D	STA #BD	
EECD	C6 B4	DEC #B4	
EECF	F0 06	BEQ \$EED7	
EED1	BA	TXA	
EED2	29 04	AND #04	
EED4	85 B5	STA #B5	
EED6	60	RTS	
EED7	A9 20	LDA #20	
EED9	2C 94 02	BIT \$0294	
EEDC	F0 14	BEQ \$EEF2	
EEDF	30 1C	BMI \$EEFC	
EEE0	70 14	BVS \$EEF6	
EEE2	A5 BD	LDA #BD	
EEE4	D0 01	BNE \$EEE7	
EEE6	CA	DEX	



EEE7	C6 B4	DEC \$B4	
EEE9	AD 93 02	LDA \$0293	
EEEC	10 E3	BPL \$EED1	
EEEE	C6 B4	DEC \$B4	
EEF0	D0 DF	BNE \$EED1	
EEF2	E6 B4	INC \$B4	
EEF4	D0 F0	BNE \$EEE6	
EEF6	A5 B0	LDA \$B0	
EEF8	F0 ED	BEQ \$EEE7	
EEFA	D0 EA	BNE \$EEE6	
EEFC	70 E9	BVS \$EEE7	
EEFE	50 E6	BVC \$EEE6	
EF00	E6 B4	INC \$B4	
EF02	A2 FF	LDX \$0FF	
EF04	D0 CB	BNE \$EED1	
EF06	AD 94 02	LDA \$0294	
EF09	4A	LSR A	
EF0A	90 07	BCC \$EF13	
EF0C	2C 01 D0	BIT \$DD01	
EF0F	10 10	BPL \$EF2E	fehlt DSR ?
EF11	50 1E	BVC \$EF31	fehlt CTS ?
EF13	A9 00	LDA \$000	
EF15	B5 B0	STA \$B0	
EF17	B5 B5	STA \$B5	
EF19	AE 98 02	LDX \$0298	
EF1C	B6 B4	STX \$B4	
EF1E	AC 9D 02	LDY \$029D	alle Bytes gesandt ?
EF21	CC 9E 02	CPY \$029E	
EF24	F0 13	BEQ \$EF39	
EF26	B1 F9	LDA (\$F9),Y	Datenbyte aus RS 232 Puffer holen
EF28	B5 B6	STA \$B6	zum senden übergeben
EF2A	EE 9D 02	INC \$029D	Pufferzeiger erhöhen
EF2D	60	RTS	
EF2E	A9 40	LDA \$040	DSR (Data Set Ready) fehlt
EF30	2C	.BYTE \$2C	
EF31	A9 10	LDA \$010	CTS (Clear To Send) fehlt
EF33	0D 97 02	ORA \$0297	Status setzen
EF36	8D 97 02	STA \$0297	
EF39	A9 01	LDA \$001	
EF3B	8D 0D D0	STA \$DD0D	IRQ für Timer A löschen
EF3E	4D A1 02	EOR \$02A1	
EF41	09 B0	ORA \$0B0	Flag für RS 232 umdrehen
EF43	8D A1 02	STA \$02A1	
EF46	8D 0D D0	STA \$DD0D	
EF49	60	RTS	
#####			Anzahl der RS 232 Datenbits berechnen
EF4A	A2 09	LDX \$009	
EF4C	A9 20	LDA \$020	
EF4E	2C 93 02	BIT \$0293	Kontrollwort
EF51	F0 01	BEQ \$EF54	
EF53	CA	DEX	
EF54	50 02	BVC \$EF58	
EF56	CA	DEX	
EF57	CA	DEX	X = Anzahl der Datenbits
EF58	60	RTS	
EF59	A6 A9	LDX \$A9	
EF5B	D0 33	BNE \$EF90	

EF5D	C6 AB	DEC \$AB	
EF5F	F0 36	BEO \$EF97	
EF61	30 0D	BMI \$EF70	
EF63	A5 A7	LDA \$A7	
EF65	45 AB	EOR \$AB	
EF67	85 AB	STA \$AB	
EF69	46 A7	LSR \$A7	
EF6B	66 AA	ROR \$AA	
EF6D	60	RTS	
EF6E	C6 AB	DEC \$AB	
EF70	A5 A7	LDA \$A7	
EF72	F0 67	BEO \$EFDB	
EF74	AD 93 02	LDA \$0293	
EF77	0A	ASL A	
EF7B	A9 01	LDA \$01	
EF7A	65 AB	ADC \$AB	
EF7C	D0 EF	BNE \$EF6D	
EF7E	A9 90	LDA \$90	
EF80	BD 0D DD	STA \$DDDD	RXD über 'Flag' empfangen
EF83	0D A1 02	ORA \$02A1	
EF86	BD A1 02	STA \$02A1	
EF89	85 A9	STA \$A9	
EF8B	A9 02	LDA \$02	
EF8D	4C 3B EF	JMP \$EF3B	IRQ für Timer B löschen
EF90	A5 A7	LDA \$A7	
EF92	D0 EA	BNE \$EF7E	
EF94	85 A9	STA \$A9	
EF96	60	RTS	
EF97	AC 9B 02	LDY \$029B	Pufferzeiger erhöhen
EF9A	CB	INY	
EF9B	CC 9C 02	CPY \$029C	
EF9E	F0 2A	BEO \$EFCA	
EFA0	BC 9B 02	STY \$029B	
EFA3	8B	DEY	
EFA4	A5 AA	LDA \$AA	:J sendendes Byte
EFA6	AE 9B 02	LDY \$029B	Anzahl Datenbits
EFA9	E0 09	CPX \$09	
EFAB	F0 04	BEO \$EFB1	
EFAE	4A	LSR A	
EFAE	EB	INX	
EFAF	D0 FB	BNE \$EFA9	
EFB1	91 F7	STA (\$F7),Y	Byte in RS 232 Puffer schreiben
EFB3	A9 20	LDA \$20	
EFB5	2C 94 02	BIT \$0294	
EFBB	F0 B4	BEO \$EF6E	
EFBA	30 B1	BMI \$EF6D	
EFBC	A5 A7	LDA \$A7	
EFBE	45 AB	EOR \$AB	
EFC0	F0 03	BEO \$EFC5	
EFC2	70 A9	BVS \$EF6D	gibt RTS
EFC4	2C	.BYTE \$2C	
EFC5	50 A6	BVC \$EF6D	gibt RTS
EFC7	A9 01	LDA \$01	Parity-Fehler
EFC9	2C	.BYTE \$2C	
ECCA	A9 04	LDA \$04	Empfangerpuffer voll
ECCC	2C	.BYTE \$2C	
ECCD	A9 B0	LDA \$B0	Break-Befehl empfangen
ECCF	2C	.BYTE \$2C	

EFD0	A9 02	LDA #002	Rahmen-Fehler
EF02	00 97 02	ORA #0297	
EF05	8D 97 02	STA #0297	Status setzen
EF0B	4C 7E EF	JMP \$EF7E	
EF0B	A5 AA	LDA \$AA	
EF0D	D0 F1	BNE \$EFD0	
EF0F	F0 EC	BEQ \$EFC0	

*****			RS 232 CKOUT, Ausgabe auf RS 232
EFE1	85 9A	STA \$9A	Geratenummer abspeichern
EFE3	AD 94 02	LDA #0294	
EFE6	4A	LSR A	
EFE7	90 29	BCC \$F012	
EFE9	A9 02	LDA #002	
EFE8	2C 01 DD	BIT \$DD01	Data Set Ready abfragen
EFEE	10 1D	BPL \$F00D	nein, dann Fehler
EFF0	D0 20	BNE \$F012	kein Request To Send ?
EFF2	AD A1 02	LDA #02A1	NMI-Flag
EFF5	29 02	AND #002	
EFF7	D0 F9	BNE \$EFF2	
EFF9	2C 01 DD	BIT \$DD01	
EFFC	70 FB	BVS \$EFF9	auf Clear To Send warten
EFFE	AD 01 DD	LDA \$DD01	
F001	09 02	ORA #002	Request To Send setzen
F003	8D 01 DD	STA \$DD01	
F006	2C 01 DD	BIT \$DD01	
F009	70 07	BVS \$F012	auf Clear To Send warten
F00B	30 F9	BMI \$F006	Data Set Ready abfragen
F00D	A9 40	LDA #0040	
F00F	8D 97 02	STA #0297	DSR Signal fehlt, Status setzen
F012	1B	CLC	
F013	60	RTS	

*****			Ausgabe in RS 232 Puffer
F014	20 28 F0	JSR \$F028	
F017	AC 9E 02	LDY #029E	
F01A	CB	INY	
F01B	CC 9D 02	CPY #029D	
F01E	F0 F4	BEQ \$F014	
F020	8C 9E 02	STY #029E	
F023	8B	DEY	
F024	A5 9E	LDA \$9E	
F026	91 F9	STA (\$F9),Y	Daten in Puffer schreiben
F028	AD A1 02	LDA #02A1	
F02B	4A	LSR A	
F02C	B0 1E	BCS \$F04C	
F02E	A9 10	LDA #0010	
F030	8D 0E DD	STA \$DD0E	
F033	AD 99 02	LDA #0299	
F036	8D 04 DD	STA \$DD04	
F039	AD 9A 02	LDA #029A	Timer für Baud-Rate neu setzen
F03C	8D 05 DD	STA \$DD05	
F03F	A9 81	LDA #0081	
F041	20 3B EF	JSR \$EF3B	Interrupt bei Unterlauf Timer A
F044	20 06 EF	JSR \$EF06	
F047	A9 11	LDA #0011	
F049	8D 0E DD	STA \$DD0E	Start Timer A, Force Load
F04C	60	RTS	

*****	RS 232 CKIN, Eingabe auf RS 232 setzen
-------	--

F04D	B5 99	STA \$99	Geratenummer
F04F	AD 94 02	LDA \$0294	
F052	4A	LSR A	
F053	90 28	BCC \$F07D	
F055	29 08	AND #\$08	
F057	F0 24	BEQ \$F07D	
F059	A9 02	LDA #\$02	
F05B	2C 01 DD	BIT \$DD01	Data Set Ready abfragen
F05E	10 AD	BPL \$F00D	nein
F060	F0 22	BEQ \$F084	
F062	AD A1 02	LDA \$02A1	
F065	4A	LSR A	
F066	B0 FA	BCS \$F062	
F068	AD 01 DD	LDA \$DD01	
F06B	29 FD	AND #\$FD	Request To Send
F06D	BD 01 DD	STA \$DD01	
F070	AD 01 DD	LDA \$DD01	
F073	29 04	AND #\$04	Data Terminal Ready
F075	F0 F9	BEQ \$F07D	nein, warten
F077	A9 90	LDA #\$90	
F079	18	CLC	
F07A	4C 3B EF	JMP \$EF3B	
F07D	AD A1 02	LDA \$02A1	
F080	29 12	AND #\$12	
F082	F0 F3	BEQ \$F077	
F084	18	CLC	
F085	60	RTS	

\*\*\*\*\*

F086	AD 97 02	LDA \$0297	GET von RS 232
F089	AC 9C 02	LDY \$029C	Status
F08C	CC 9B 02	CPY \$029B	Pufferzeiger
F08F	F0 08	BEQ \$F09C	gleich
F091	29 F7	AND #\$F7	Puffer leer ?
F093	BD 97 02	STA \$0297	im Status löschen
F096	B1 F7	LDA (\$F7),Y	Byte aus Puffer holen
F098	EE 9C 02	INC \$029C	Pufferzeiger erhöhen
F09B	60	RTS	

F09C	09 08	ORA #\$08	Puffer leer
F09E	BD 97 02	STA \$0297	Status setzen
F0A1	A9 00	LDA #\$00	Null übergeben
F0A3	60	RTS	

\*\*\*\*\*

F0A4	4B	PHA	RS 232
F0A5	AD A1 02	LDA \$02A1	Flag gesetzt ?
F0AB	F0 11	BEQ \$F0BB	nein, dann fertig
F0AA	AD A1 02	LDA \$02A1	
F0AD	29 03	AND #\$03	
F0AF	D0 F9	BNE \$F0AA	auf Bit 0 und 1 warten
F0B1	A9 10	LDA #\$10	
F0B3	BD 0D DD	STA \$DD0D	Interrupt über 'Flag'-Leitung
F0B6	A9 00	LDA #\$00	
F0B8	BD A1 02	STA \$02A1	Flag zurücksetzen
F0BB	68	PLA	
F0BC	60	RTS	

\*\*\*\*\*

F0BD	0D 49 2F 4F 20 45 52 4F	Systemmeldungen
		i/o error #

F0C6	52 20 A3	
F0C9	0D 53 45 41 52 43 48 49	searching
F0D1	4E 47 A0	
F0D4	46 4F 52 A0	for
F0DB	0D 50 52 45 53 53 20 50	press play on tape
F0D9	4C 41 59 20 4F 4E 20 54	
F0DB	41 50 C5	
F0EB	50 52 45 53 53 20 52 45	press record & play on tape
F0F3	43 4F 52 44 20 26 20 50	
F0FB	4C 41 59 20 4F 4E 20 54	
F103	41 50 C5	
F106	0D 4C 4F 41 44 49 4E C7	loading
F10E	0D 53 41 56 49 4E 47 A0	saving
F116	0D 56 45 52 49 46 59 49	verifying
F11E	4E C7	
F120	0D 46 4F 55 4E 44 A0	found
F127	0D 4F 4B BD	ok

*****		Systemmeldungen ausgeben
F12B	24 9D BIT \$9D	Direkt-Modus Flag
F12D	10 0D BPL \$F13C	Programm, dann überspringen
F12F	B9 BD F0 LDA \$F0BD,Y	Offset der Meldung in Y
F132	0B PHP	
F133	29 7F AND #\$7F	Bit 7 löschen
F135	20 D2 FF JSR \$FFD2	ausgeben
F13B	C8 INY	
F139	28 PLP	
F13A	10 F3 BPL \$F12F	noch weitere Buchstaben ?
F13C	1B CLC	
F13D	60 RTS	

*****		GETIN
F13E	A5 99 LDA \$99	Eingabegerät
F140	D0 0B BNE \$F14A	
F142	A5 C6 LDA \$C6	Anzahl der Zeichen im Tastaturpuffer
F144	F0 0F BEQ \$F155	kein Zeichen ?
F146	78 SEI	
F147	4C B4 E5 JMP \$E5B4	Zeichen aus Tastaturpuffer holen
F14A	C9 02 CMP #\$02	
F14C	D0 1B BNE \$F166	nicht RS 232, dann zur BASIN-Routine
F14E	B4 97 STY \$97	
F150	20 B6 F0 JSR \$F0B6	Get von RS 232
F153	A4 97 LDY \$97	
F155	1B CLC	
F156	60 RTS	

*****		BASIN Eingabe eines Zeichens
F157	A5 99 LDA \$99	Geratenummer
F159	D0 0B BNE \$F166	nicht Tastatur ?
F15B	A5 D3 LDA \$D3	Cursorposition
F15D	B5 CA STA \$CA	
F15F	A5 D6 LDA \$D6	für Tastatureingabe setzen
F161	B5 C9 STA \$C9	
F163	4C 32 E6 JMP \$E632	Eingabe vom Bildschirm

F166	C9 03 CMP #\$03	
F168	D0 09 BNE \$F173	
*****		vom Bildschirm
F16A	B5 D0 STA \$D0	
F16C	A5 D5 LDA \$D5	

F16E	B5 C8	STA \$C8	
F170	4C 32 E6	JMP \$E632	
F173	B0 38	BCS \$F1AD	vom IEC-Bus
F175	C9 02	CMP #\$02	
F177	F0 3F	BEQ \$F1B8	
*****			vom Band
F179	B6 97	STX \$97	X-Register merken
F17B	20 99 F1	JSR \$F199	ein Zeichen vom Band holen
F17E	B0 16	BCS \$F196	
F180	48	PHA	
F181	20 99 F1	JSR \$F199	ein Zeichen vom Band holen
F184	B0 0D	BCS \$F193	Fehler ?
F186	D0 05	BNE \$F18D	letztes Zeichen ?
F188	A9 40	LDA #\$40	EOF
F18A	20 1C FE	JSR \$FE1C	Status setzen
F18D	C6 A6	DEC \$A6	Bandpuffer Zeiger erniedrigen
F18F	A6 97	LDX \$97	X-Register zurückholen
F191	6B	PLA	
F192	60	RTS	
F193	AA	TAX	Fehlernummer nach X
F194	6B	PLA	Zeichen zurückholen
F195	BA	TXA	Fehlernummer in Akku
F196	A6 97	LDX \$97	X-Register zurückholen
F198	60	RTS	
*****			ein Zeichen vom Band holen
F199	20 0D FB	JSR \$FB0D	Bandpuffer Zeiger erhöhen
F19C	D0 0B	BNE \$F1A9	noch Zeichen im Puffer
F19E	20 41 FB	JSR \$FB41	nein, nächsten Block vom Band holen
F1A1	B0 11	BCS \$F1B4	STOP-Taste, dann Abbruch
F1A3	A9 00	LDA #\$00	
F1A5	B5 A6	STA \$A6	Pufferzeiger auf null, Zeichen holen
F1A7	F0 F0	BEQ \$F199	
F1A9	B1 B2	LDA (\$B2),Y	Zeichen aus Puffer lesen
F1AB	1B	CLC	
F1AC	60	RTS	
F1AD	A5 90	LDA \$90	Status testen
F1AF	F0 04	BEQ \$F1B5	ok
F1B1	A9 0D	LDA #\$0D	'CR' Kode ausgeben
F1B3	1B	CLC	
F1B4	60	RTS	
*****			IEC-Eingabe
F1B5	4C 13 EE	JMP \$EE13	ein Byte vom IEC-Bus holen
*****			RS 232 Eingabe
F1B8	20 4E F1	JSR \$F14E	ein Byte von RS 232 holen
F1BB	B0 F7	BCS \$F1B4	Fehler ?
F1BD	C9 00	CMP #\$00	Nullbyte ?
F1BF	D0 F2	BNE \$F1B3	nein, dann ok
F1C1	AD 97 02	LDA \$0297	Status
F1C4	29 60	AND #\$60	fehlt DSR ?
F1C6	D0 E9	BNE \$F1B1	ja, 'CR' zurückgeben
F1C8	F0 EE	BEQ \$F1B8	nein, neuer Versuch
*****			BSOUT Ausgabe eines Zeichens
F1CA	4B	PHA	
F1CB	A5 9A	LDA \$9A	Gerätenummer für Ausgabe
F1CD	C9 03	CMP #\$03	Bildschirm ?

F1CF	D0 04	BNE \$F1D5	nein
F1D1	68	PLA	
F1D2	4C 16 E7	JMP \$E716	ein Zeichen auf Bildschirm ausgeben
F1D5	90 04	BCC \$F1D8	
F1D7	68	PLA	
F1D8	4C DD ED	JMP \$EDDD	ein Byte auf IEC-Bus ausgeben
F1D8	4A	LSR A	
F1DC	68	PLA	
F1DD	85 9E	STA \$9E	auszugebendes Zeichen merken
F1DF	8A	TXA	
F1E0	4B	PHA	
F1E1	9B	TYA	
F1E2	4B	PHA	
F1E3	90 23	BCC \$F208	RS 232 Ausgabe
F1E5	20 0D FB	JSR \$FB0D	Bandpuffer Zeiger erhöhen
F1E8	D0 0E	BNE \$F1F8	Puffer voll ?
F1EA	20 64 FB	JSR \$FB64	Puffer auf Band schreiben
F1ED	B0 0E	BCS \$F1FD	STOP-Taste, dann Abbruch
F1EF	A9 02	LDA #\$02	Kontrollbyte für Datenblock
F1F1	A0 00	LDY #\$00	
F1F3	91 B2	STA (\$B2),Y	in Puffer schreiben
F1F5	CB	INY	
F1F6	84 A6	STY \$A6	Pufferzeiger erhöhen und merken
F1F8	A5 9E	LDA \$9E	
F1FA	91 B2	STA (\$B2),Y	Zeichen in Puffer schreiben
F1FC	1B	CLC	
F1FD	68	PLA	
F1FE	AB	TAY	
F1FF	68	PLA	
F200	AA	TAX	
F201	A5 9E	LDA \$9E	Zeichen zurückholen
F203	90 02	BCC \$F207	ok ?
F205	A9 00	LDA #\$00	Flag für 'STOP-Taste gedrückt'
F207	60	RTS	
*****			RS 232 Ausgabe
F208	20 17 F0	JSR \$F017	ein Zeichen in RS 232 Puffer schreiben
F20B	4C FC F1	JMP \$F1FC	
*****			CHKIN Eingabegerät setzen
F20E	20 0F F3	JSR \$F30F	sucht logische Filenummer
F211	F0 03	BEQ \$F216	gefunden ?
F213	4C 01 F7	JMP \$F701	'file not open'
F216	20 1F F3	JSR \$F31F	setzt Fileparameter
F219	A5 BA	LDA \$BA	Gerätenummer
F21B	F0 16	BEQ \$F233	0, Tastatur
F21D	C9 03	CMR #\$03	
F21F	F0 12	BEQ \$F233	3, Bildschirm
F221	B0 14	BCS \$F237	IEC-Bus
F223	C9 02	CMR #\$02	RS 232 ?
F225	D0 03	BNE \$F22A	nein, dann Band
F227	4C 4D F0	JMP \$F04D	ja
*****			Band als Eingabegerät setzen
F22A	A6 B9	LDX \$B9	Sekundäradresse
F22C	E0 60	CPX #\$60	null ?
F22E	F0 03	BEQ \$F233	
F230	4C 0A F7	JMP \$F70A	'not input file'
F233	85 99	STA \$99	Gerätenummer für Ausgabe
F235	1B	CLC	

F236	60	RTS	
F237	AA	TAX	
F238	20 09 ED	JSR \$ED09	TALK senden
F23B	A5 B9	LDA \$B9	Sekundaradresse
F23D	10 06	BPL \$F245	
F23F	20 CC ED	JSR \$EDCC	wartet auf Talk-Signal
F242	4C 4B F2	JMP \$F24B	
F245	20 C7 ED	JSR \$EDC7	Sekundaradresse für TALK senden
F24B	8A	TXA	
F249	24 90	BIT \$90	Status
F24B	10 E6	BPL \$F233	Ok ?
F24D	4C 07 F7	JMP \$F707	'device not present'
*****			
F250	20 0F F3	JSR \$F30F	CKOUT Ausgabegerät setzen
F253	F0 03	BEQ \$F25B	sucht logische Filenummer
F255	4C 01 F7	JMP \$F701	gefunden
F25B	20 1F F3	JSR \$F31F	'file not open'
F25B	A5 BA	LDA \$BA	setzt Fileparameter
F25D	D0 03	BNE \$F262	Gerätenummer
F25F	4C 0D F7	JMP \$F70D	ungleich null ?
F262	C9 03	CMP \$03	'not input file'
F264	F0 0F	BEQ \$F275	Bildschirm ?
F266	B0 11	BCS \$F279	ja
F268	C9 02	CMP \$02	IEC-Bus
F26A	D0 03	BNE \$F26F	RS 232 ?
F26C	4C E1 EF	JMP \$EFE1	
*****			
F26F	A6 B9	LDX \$B9	Band als Ausgabegerät setzen
F271	E0 60	CPI \$60	Sekundaradresse
F273	F0 EA	BEQ \$F25F	null ?
F275	85 9A	STA \$9A	Bandfile zum lesen, 'not output file'
F277	1B	CLC	Nummer des Ausgabegeräts setzen
F27B	60	RTS	
*****			
F279	AA	TAX	Ausgabe auf IEC-Bus legen
F27A	20 0C ED	JSR \$ED0C	
F27D	A5 B9	LDA \$B9	LISTEN senden
F27F	10 05	BPL \$F286	Sekundaradresse
F281	20 BE ED	JSR \$EDBE	
F284	D0 03	BNE \$F289	ATN zurücksetzen
F286	20 B9 ED	JSR \$EDB9	
F289	8A	TXA	Sekundaradresse für LISTEN senden
F28A	24 90	BIT \$90	
F28C	10 E7	BPL \$F275	Status
F28E	4C 07 F7	JMP \$F707	Ok
*****			
F291	20 14 F3	JSR \$F314	'device not present'
F294	F0 02	BEQ \$F29B	
F296	1B	CLC	CLOSE logische Filenummer in A
F297	60	RTS	sucht logische Filenummer
F298	20 1F F3	JSR \$F31F	File nicht vorhanden, dann fertig
F29B	8A	TXA	
F29C	4B	PHA	Fileparameter setzen
F29D	A5 BA	LDA \$BA	
F29F	F0 50	BEQ \$F2F1	Geräteadresse
F2A1	C9 03	CMP \$03	Tastatur ?



F2A3	F0 4C	BEQ \$F2F1	Bildschirm ?
F2A5	B9 47	BCS \$F2EE	IEC ?
F2A7	C9 02	CMP #02	RS 232 ?
F2A9	D9 1D	BNE \$F2CB	Band
*****			RS 232 File schließen
F2AB	68	PLA	
F2AC	20 F2 F2	JSR \$F2F2	Fileeintrag in Tabelle löschen
F2AF	20 83 F4	JSR \$F4B3	CIAs für I/O rücksetzen
F2B2	20 27 FE	JSR \$FE27	Memory-Top holen
F2B5	A5 FB	LDA \$FB	Eingabepuffer
F2B7	F0 01	BEQ \$F2BA	
F2B9	CB	INY	
F2BA	A5 FA	LDA \$FA	Ausgabepuffer
F2BC	F0 01	BEQ \$F2BF	
F2BE	CB	INY	
F2BF	A9 00	LDA #00	
F2C1	B5 FB	STA \$FB	Puffer freigeben
F2C3	B5 FA	STA \$FA	
F2C5	4C 7D F4	JMP \$F47D	Memory Top neu setzen
*****			Band File schließen
F2CB	A5 B9	LDA \$B9	Sekundäradresse
F2CA	29 0F	AND #0F	
F2CC	F0 23	BEQ \$F2F1	File zum Lesen ?
F2CE	20 D0 F7	JSR \$F7D0	Band-Puffer Startadresse holen
F2D1	A9 00	LDA #00	
F2D3	78	SEC	
F2D4	20 DD F1	JSR \$F1DD	
F2D7	20 64 FB	JSR \$FB64	Puffer auf Band schreiben
F2DA	90 04	BCC \$F2E0	
F2DC	68	PLA	
F2DD	A9 00	LDA #00	
F2DF	60	RTS	
F2E0	A5 B9	LDA \$B9	Sekundäradresse
F2E2	C9 62	CMP #62	gleich 2
F2E4	D0 0B	BNE \$F2F1	
F2E6	A9 05	LDA #05	Kontrollbyte für EOT-Header
F2E8	20 6A F7	JSR \$F76A	Block auf Band schreiben
F2E9	4C F1 F2	JMP \$F2F1	
F2EE	20 42 F6	JSR \$F642	IEC-File schließen
F2F1	68	PLA	
F2F2	AA	TAX	
F2F3	C6 98	DEC \$98	Anzahl der offenen Files erniedrigen
F2F5	E4 98	CPI \$98	
F2F7	F0 14	BEQ \$F30D	gleich null, dann fertig
F2F9	A4 98	LDY \$98	
F2FB	B9 59 02	LDA \$0259,Y	
F2FE	9D 59 02	STA \$0259,X	
F301	B9 63 02	LDA \$0263,Y	Fileparameter-Tabelle aufschließen
F304	9D 63 02	STA \$0263,X	
F307	B9 6D 02	LDA \$026D,Y	
F30A	9D 6D 02	STA \$026D,X	
F30D	18	CLC	
F30E	60	RTS	
*****			sucht logische Filenummer (in X)
F30F	A9 00	LDA #00	
F311	B5 90	STA \$90	Status löschen
F313	BA	TXA	

F314	A6 98	LDI \$98	Anzahl der offenen Files
F316	CA	DEX	
F317	30 15	BMI \$F32E	
F319	DD 59 02	CMP \$0259,X	sucht Eintrag in Tabelle
F31C	D0 F8	BNE \$F316	
F31E	60	RTS	
*****			setzt Fileparameter
F31F	BD 59 02	LDA \$0259,X	
F322	B5 B8	STA \$B8	logische Filenummer
F324	BD 63 02	LDA \$0263,X	
F327	B5 BA	STA \$BA	Geräteadresse
F329	BD 6D 02	LDA \$026D,X	
F32C	B5 B9	STA \$B9	Sekundäradresse
F32E	60	RTS	
*****			CLALL schließt alle Ein-/Ausgabe Kanäle
F32F	A9 00	LDA #\$00	
F331	B5 98	STA \$98	Anzahl der offenen Files gleich null
*****			CLRCH schließt aktiven I/O-Kanal
F333	A2 03	LDX #\$03	
F335	E4 9A	CPX \$9A	Nummer des Ausgabegeräts
F337	B0 03	BCS \$F33C	kleiner als 3
F339	20 FE ED	JSR \$EDFE	IEC, UNLISTEN senden
F33C	E4 99	CPX \$99	Nummer des Eingabegeräts
F33E	B0 03	BCS \$F343	kleiner als 3
F340	20 EF ED	JSR \$EDEF	IEC, UNTALK senden
F343	B6 9A	STX \$9A	Ausgabe wieder auf Bildschirm
F345	A9 00	LDA #\$00	
F347	B5 99	STA \$99	Eingabe wieder von Tastatur
F349	60	RTS	
*****			OPEN
F34A	A6 B8	LDX \$B8	Filenummer
F34C	D0 03	BNE \$F351	ungleich null
F34E	4C 0A F7	JMP \$F70A	'not input file' (??)
F351	20 0F F3	JSR \$F30F	sucht logische Filenummer
F354	D0 03	BNE \$F359	nicht gefunden, kann neu angelegt werden
F356	4C FE F6	JMP \$F6FE	'file open'
F359	A6 98	LDX \$98	Anzahl der offenen Files
F35B	E0 0A	CPX #\$0A	mit 10 vergleichen
F35D	90 03	BCC \$F362	
F35F	4C FB F6	JMP \$F6FB	'too many files'
F362	E6 98	INC \$98	Anzahl erhöhen
F364	A5 B8	LDA \$B8	logische Filenummer
F366	9D 59 02	STA \$0259,X	
F369	A5 B9	LDA \$B9	Sekundäradresse
F36B	09 60	ORA #\$60	
F36D	B5 B9	STA \$B9	
F36F	9D 6D 02	STA \$026D,X	
F372	A5 BA	LDA \$BA	Gerätenummer
F374	9D 63 02	STA \$0263,X	in die entsprechenden Tabellen schreiben
F377	F0 5A	BEQ \$F3D3	Tastatur ?
F379	C9 03	CMP #\$03	
F37B	F0 56	BEQ \$F3D3	Bildschirm
F37D	90 05	BCC \$F384	
F37F	20 D5 F3	JSR \$F3D5	File auf IEC-Bus eröffnen
F382	90 4F	BCC \$F3D3	
F384	C9 02	CMP #\$02	Band ?

F3B6	D0 03	BNE \$F3B8	nein
F3B8	4C 09 F4	JMP \$F409	RS 232 open
F3B8	20 D0 F7	JSR \$F7D0	Bandpuffer Startadresse holen
F3BE	B0 03	BCS \$F393	
F390	4C 13 F7	JMP \$F713	'illegal device number'
F393	A5 B9	LDA \$B9	Sekundäradresse
F395	29 0F	AND #\$0F	
F397	D0 1F	BNE \$F3B8	ungleich null dann schreiben
F399	20 17 F8	JSR \$F817	wartet auf Play-Taste
F39C	B0 36	BCS \$F3D4	STOP-Taste gedrückt ?
F39E	20 AF F5	JSR \$F5AF	'searching' ('for name') ausgeben
F3A1	A5 B7	LDA \$B7	Länge des Filenamens
F3A3	F0 0A	BEQ \$F3AF	kein Filename, dann weiter
F3A5	20 EA F7	JSR \$F7EA	sucht gewünschten Bandheader
F3AB	90 18	BCC \$F3C2	gefunden
F3AA	F0 28	BEQ \$F3D4	ok
F3AC	4C 04 F7	JMP \$F704	EOT, 'file not found' ausgeben
F3AF	20 2C F7	JSR \$F72C	nächsten Bandheader suchen
F3B2	F0 20	BEQ \$F3D4	EOT, Fehler
F3B4	90 0C	BCC \$F3C2	gefunden
F3B6	B0 F4	BCS \$F3AC	PRG-File, weiter suchen
F3B8	20 38 F8	JSR \$F838	wartet auf Record & Play Taste
F3BB	B0 17	BCS \$F3D4	STOP-Taste, dann Abbruch
F3BD	A9 04	LDA #\$04	Kontrollbyte für Datenheader
F3BF	20 6A F7	JSR \$F76A	Header auf Band schreiben
F3C2	A9 BF	LDA #\$BF	Zeiger auf Ende des Bandpuffers
F3C4	A4 B9	LDY \$B9	Sekundäradresse
F3C6	C0 60	CPY #\$60	
F3C8	F0 07	BEQ \$F3D1	gleich null, dann weiter
F3CA	A0 00	LDY #\$00	
F3CC	A9 02	LDA #\$02	Kontrollbyte für Datenblock
F3CE	91 B2	STA (\$B2),Y	in Bandpuffer schreiben
F3D0	98	TYA	
F3D1	B5 A6	STA \$A6	Zeiger in Bandpuffer
F3D3	18	CLC	
F3D4	60	RTS	

*****			File auf IEC-Bus eröffnen
F3D5	A5 B9	LDA \$B9	Sekundäradresse
F3D7	30 FA	BM1 \$F3D3	
F3D9	A4 B7	LDY \$B7	Länge des Filenamens
F3DB	F0 F6	BEQ \$F3D3	gleich null, dann fertig
F3DD	A9 00	LDA #\$00	
F3DF	B5 90	STA \$90	Status löschen
F3E1	A5 BA	LDA \$BA	Geräteadresse
F3E3	20 0C ED	JSR \$ED0C	LISTEN
F3E6	A5 B9	LDA \$B9	Sekundäradresse
F3E8	09 F0	ORA #\$F0	
F3EA	20 B9 ED	JSR \$EDB9	senden
F3ED	A5 90	LDA \$90	Status testen
F3EF	10 05	BPL \$F3F6	ok
F3F1	68	PLA	
F3F2	68	PLA	
F3F3	4C 07 F7	JMP \$F707	'device not present'
F3F6	A5 B7	LDA \$B7	Länge des Filenamens
F3F8	F0 0C	BEQ \$F406	
F3FA	A0 00	LDY #\$00	
F3FC	B1 BB	LDA (\$BB),Y	Filename
F3FE	20 DD ED	JSR \$EDDD	auf IEC-Bus ausgeben
F401	C8	INY	

F402	C4 B7	CPY #B7	
F404	D0 F6	BNE \$F3FC	
F406	4C 54 F6	JMP \$F654	UNLISTEN, return
***** RS 232 Open			
F409	20 B3 F4	JSR \$F4B3	CIAs setzen
F40C	BC 97 02	STY \$0297	RS 232 Status löschen
F40F	C4 B7	CPY #B7	Länge des "Filenames"
F411	F0 0A	BEQ \$F41D	
F413	B1 BB	LDA (\$BB),Y	die ersten 4 Zeichen speichern
F415	99 93 02	STA \$0293,Y	
F418	EB	INY	
F419	C0 04	CPY #04	
F41B	D0 F2	BNE \$F40F	
F41D	20 4A EF	JSR \$EF4A	Anzahl der Datenbits berechnen
F420	BE 98 02	STX \$0298	und speichern
F423	AD 93 02	LDA \$0293	Kontrollregister
F426	29 0F	AND #0F	Bits für Baud-rate isolieren
F428	F0 1C	BEQ \$F446	
F42A	0A	ASL A	* 2 für Tabelle
F42B	AA	TAX	
F42C	AD A6 02	LDA \$02A6	NTSC-Version ?
F42F	D0 09	BNE \$F43A	nein
F431	BC C1 FE	LDY \$FEC1,X	Baud-Rate, High für NTSC-Timing
F434	BD C0 FE	LDA \$FEC0,X	Baud-Rate, Low
F437	4C 40 F4	JMP \$F440	
F43A	BC EB E4	LDY \$E4EB,X	Baud-Rate, High für PAL-Timing
F43D	BD EA E4	LDA \$E4EA,X	Baud-Rate, Low
F440	BC 96 02	STY \$0296	
F443	BD 95 02	STA \$0295	speichern
F446	AD 95 02	LDA \$0295	
F449	0A	ASL A	
F44A	20 2E FF	JSR \$FF2E	Kode für Baud-Rate ermitteln
F44D	AD 94 02	LDA \$0294	
F450	4A	LSR A	
F451	90 09	BCC \$F45C	
F453	AD 01 DD	LDA \$DD01	fehlt DSR ?
F456	0A	ASL A	
F457	B0 03	BCC \$F45C	nein
F459	20 0D F0	JSR \$F00D	Status für DSR setzen
F45C	AD 9B 02	LDA \$029B	
F45F	BD 9C 02	STA \$029C	Pufferzeiger für RS 232 Ein- und Ausgabe setzen
F462	AD 9E 02	LDA \$029E	
F465	BD 9D 02	STA \$029D	
F468	20 27 FE	JSR \$FE27	Memory Top holen
F46B	A5 F8	LDA \$F8	
F46D	D0 05	BNE \$F474	Eingabepuffer bereits angelegt ?
F46F	8B	DEY	
F470	B4 FB	STY \$F8	Zeiger für RS 232 Eingabepuffer
F472	B6 F7	STX \$F7	
F474	A5 FA	LDA \$FA	
F476	D0 05	BNE \$F47D	Ausgabepuffer bereits angelegt ?
F478	8B	DEY	
F479	B4 FA	STY \$FA	Zeiger für RS 232 Ausgabepuffer
F47B	B6 F9	STX \$F9	
F47D	38	SEC	
F47E	A9 F0	LDA #F0	Flag setzen
F480	4C 2D FE	JMP \$FE2D	Memory-Top neu setzen
***** CIAs nach RS 232 rücksetzen			

```

F483  A9 7F      LDA #$7F
F485  BD 00 DD    STA $DD0D
F488  A9 06      LDA #$06
F48A  BD 03 DD    STA $DD03
F48D  BD 01 DD    STA $DD01
F490  A9 04      LDA #$04
F492  0D 00 DD    ORA $DD00
F495  BD 00 DD    STA $DD00
F498  A0 00      LDY #$00
F49A  BC A1 02    STY $02A1
F49D  60          RTS

```

IRQ's rücksetzen  
 Bit 1 und 2 Ausgang  
 PORT B Richtung  
 PORT A Richtung

Bit 2 = TXD

NMI-Flag löschen

\*\*\*\*\*

```

F49E  B6 C3      STX $C3
F4A0  B4 C4      STY $C4
F4A2  6C 30 03   JMP ($0330)
F4A5  B5 93      STA $93
F4A7  A9 00      LDA #$00
F4A9  B5 90      STA $90
F4AB  A5 BA      LDA $BA
F4AD  D0 03      BNE $F4B2
F4AF  4C 13 F7   JMP $F713
F4B2  C9 03      CMP #$03
F4B4  F0 F9      BEQ $F4AF
F4B6  90 7B      BCC $F533

```

LOAD - Routine

Startadresse speichern

JMP \$F4A5      LOAD-Vektor  
 Load/Verify Flag

Status löschen

Geräte-Adresse

ungleich null, dann weiter

'illegal device number'

Bildschirm ?

ja, Fehler

kleiner 3, dann vom Band

\*\*\*\*\*

```

F4B8  A4 B7      LDY $B7
F4BA  D0 03      BNE $F4BF
F4BC  4C 10 F7   JMP $F710
F4BF  A6 B9      LDX $B9
F4C1  20 AF F5   JSR $F5AF
F4C4  A9 60      LDA #$60
F4C6  B5 B9      STA $B9
F4C8  20 D5 F3   JSR $F3D5
F4CB  A5 BA      LDA $BA
F4CD  20 09 ED   JSR $ED09
F4D0  A5 B9      LDA $B9
F4D2  20 C7 ED   JSR $EDC7
F4D5  20 13 EE   JSR $EE13
F4D8  B5 AE      STA $AE
F4DA  A5 90      LDA $90
F4DC  4A         LSR A
F4DD  4A         LSR A
F4DE  B0 50      BCS $F530
F4E0  20 13 EE   JSR $EE13
F4E3  B5 AF      STA $AF
F4E5  BA         TXA
F4E6  D0 08      BNE $F4F0
F4E8  A5 C3      LDA $C3
F4EA  B5 AE      STA $AE
F4EC  A5 C4      LDA $C4
F4EE  B5 AF      STA $AF
F4F0  20 D2 F5   JSR $F5D2
F4F3  A9 FD      LDA $FD
F4F5  25 90      AND $90
F4F7  B5 90      STA $90
F4F9  20 E1 FF   JSR $FFE1
F4FC  D0 03      BNE $F501
F4FE  4C 33 F6   JMP $F633

```

IEC-Load

Länge des Filenamens

ungleich null, dann ok

'missing filename'

Sekundaradresse

'searching for filename'

Sekundaradresse null

File auf IEC-Bus eröffnen

Gerätenummer

TALK senden

Sekundaradresse senden

Byte vom IEC-Bus holen

als Startadresse low speichern

Status

Time out, dann Fehler

Startadresse high holen

Sekundaradresse ungleich null ?

nein, dann ab vorgegebener Adresse  
 laden

'loading'/'verifying' ausgeben

Time-out Bit löschen

Stop-Taste abfragen

nicht gedrückt, dann weiter

File schließen

F501	20 13 EE	JSR \$EE13	Programbyte vom Bus holen
F504	AA	TAX	
F505	A5 90	LDA \$90	Status testen
F507	4A	LSR A	
F508	4A	LSR A	
F509	B0 E8	BCS \$F4F3	Fehler, dann abbrechen
F50B	8A	TXA	
F50C	A4 93	LDY \$93	Load/Verify Flag testen
F50E	F0 0C	BEQ \$F51C	gleich null, dann LOAD
F510	A0 00	LDY \$00	
F512	D1 AE	CMP (\$AE),Y	Verify, Vergleich
F514	F0 08	BEQ \$F51E	
F516	A9 10	LDA \$010	ungleich, dann Status setzen
F518	20 1C FE	JSR \$FE1C	Status setzen
F51B	2C	.BYTE \$2C	
F51C	91 AE	STA (\$AE),Y	Byte abspeichern
F51E	E6 AE	INC \$AE	
F520	D0 02	BNE \$F524	Adresse erhöhen
F522	E6 AF	INC \$AF	
F524	24 90	BIT \$90	Status
F526	50 CB	BVC \$F4F3	noch kein EOF ?
F528	20 EF ED	JSR \$EDEF	UNTALK senden
F52B	20 42 F6	JSR \$F642	File schliessen
F52E	90 79	BCC \$F5A9	kein Fehler ?
F530	4C 04 F7	JMP \$F704	'file not found'
*****			
F533	4A	LSR A	Gerätenummer
F534	B0 03	BCS \$F539	eins (Band) , dann weiter
F536	4C 13 F7	JMP \$F713	RS 232, 'illegal device number'
F539	20 D0 F7	JSR \$F7D0	Bandpuffer Startadresse holen
F53C	B0 03	BCS \$F541	
F53E	4C 13 F7	JMP \$F713	'illegal device number'
F541	20 17 F8	JSR \$F817	wartet auf Play-Taste
F544	B0 68	BCS \$F5AE	STOP-Taste, dann Abbruch
F546	20 AF F5	JSR \$F5AF	'searching' ( 'for name' ) ausgeben
F549	A5 87	LDA \$B7	Länge des Filenamens
F54B	F0 09	BEQ \$F556	gleich 0, dann weiter
F54D	20 EA F7	JSR \$F7EA	gewünschten Bandheader suchen
F550	90 08	BCC \$F55D	gefunden
F552	F0 5A	BEQ \$F5AE	Abbruch
F554	B0 DA	BCS \$F530	EOT, dann 'file not found'
F556	20 2C F7	JSR \$F72C	nächsten Bandheader suchen
F559	F0 53	BEQ \$F5AE	Abbruch
F55B	B0 D3	BCS \$F530	'EOT', dann 'file not found'
F55D	A5 90	LDA \$90	Status holen
F55F	29 10	AND \$010	EOF-Bit ausblenden
F561	38	SEC	
F562	D0 4A	BNE \$F5AE	anderer Fehler ?
F564	E0 01	CPX \$01	Header-Typ 1 = BASIC-Programm (verschieblich)
F566	F0 11	BEQ \$F579	
F568	E0 03	CPX \$03	3 = Maschinen-Programm (absolut)
F56A	D0 DD	BNE \$F549	
F56C	A0 01	LDY \$01	
F56E	B1 B2	LDA (\$B2),Y	Startadresse low
F570	B5 C3	STA \$C3	
F572	C8	INY	
F573	B1 B2	LDA (\$B2),Y	Startadresse high
F575	B5 C4	STA \$C4	
F577	B0 04	BCS \$F57D	

F579	A5 B9	LDA \$B9	Sekundär-Adresse
F57B	D0 EF	BNE \$F56C	ungleich null, dann nicht verschieblich laden
F57D	A0 03	LDY #03	
F57F	B1 B2	LDA (\$B2),Y	
F581	A0 01	LDY #01	Endadresse minus
F583	F1 B2	SBC (\$B2),Y	
F585	AA	TAX	
F586	A0 04	LDY #04	
F588	B1 B2	LDA (\$B2),Y	
F58A	A0 02	LDY #02	Startadresse
F58C	F1 B2	SBC (\$B2),Y	
F58E	A8	TAY	
F58F	18	CLC	gleich Programmlänge
F590	BA	TXA	
F591	65 C3	ADC \$C3	
F593	B5 AE	STA \$AE	Programmlänge + Startadresse
F595	98	TYA	
F596	65 C4	ADC \$C4	gleich Endadresse
F598	B5 AF	STA \$AF	
F59A	A5 C3	LDA \$C3	
F59C	B5 C1	STA \$C1	Startadresse nach \$C1/\$C2
F59E	A5 C4	LDA \$C4	
F5A0	B5 C2	STA \$C2	
F5A2	20 D2 F5	JSR \$F5D2	'loading' / 'verifying' ausgeben
F5A5	20 4A F8	JSR \$F84A	Programm vom Band laden
F5A8	24	.BYTE \$24	
F5A9	18	CLC	
F5AA	A6 AE	LDX \$AE	Endadresse nach X/Y
F5AC	A4 AF	LDY \$AF	
F5AE	60	RTS	

*****			'searching' (for filename) ausgeben
F5AF	A5 9D	LDA \$9D	Direkt-Modus testen
F5B1	10 1E	BPL \$F5D1	nein, dann übergehen
F5B3	A0 0C	LDY #0C	Offset für 'searching'
F5B5	20 2F F1	JSR \$F12F	Meldung ausgeben
F5B8	A5 B7	LDA \$B7	Länge des Filenames
F5BA	F0 15	BEQ \$F5D1	gleich null, dann fertig
F5BC	A0 17	LDY #17	Offset für 'for'
F5BE	20 2F F1	JSR \$F12F	Meldung ausgeben
F5C1	A4 B7	LDY \$B7	Länge des Filenames
F5C3	F0 0C	BEQ \$F5D1	gleich null, dann fertig
F5C5	A0 00	LDY #00	
F5C7	B1 BB	LDA (\$BB),Y	Filenames holen
F5C9	20 D2 FF	JSR \$FFD2	und ausgeben
F5CC	CB	INY	
F5CD	C4 B7	CPY \$B7	
F5CF	D0 F6	BNE \$F5C7	
F5D1	60	RTS	

*****			'loading/verifying' ausgeben
F5D2	A0 49	LDY #49	Offset für 'loading'
F5D4	A5 93	LDA \$93	Load/Verify-Flag testen
F5D6	F0 02	BEQ \$F5DA	Load, dann ausgeben
F5D8	A0 59	LDY #59	Offset für 'verifying'
F5DA	4C 2B F1	JMP \$F12B	Meldung ausgeben

*****			SAVE - Routine
F5DD	B6 AE	STX \$AE	Endadresse low
F5DF	B4 AF	STY \$AF	" high

F5E1	AA	TAX	
F5E2	B5 00	LDA #00,X	
F5E4	B5 C1	STA \$C1	Startadresse low
F5E6	B5 01	LDA #01,X	
F5E8	B5 C2	STA \$C2	Startadresse high
F5EA	4C 32 03	JMP (\$0332)	SAVE-Vektor, JMP #F5ED
F5ED	A5 BA	LDA \$BA	Geräteadresse
F5EF	D0 03	BNE \$F5F4	
F5F1	4C 13 F7	JMP \$F713	'illegal device number'
F5F4	C9 03	CMR #03	
F5F6	F0 F9	BEQ \$F5F1	Bildschirm, Fehler
F5FB	90 5F	BCC \$F659	Band
*****			
F5FA	A9 61	LDA #61	Speichern auf IEC-Bus
F5FC	B5 B9	STA \$B9	Sekundär-Adresse 1
F5FE	A4 B7	LDY \$B7	setzen
F600	D0 03	BNE \$F605	Länge des Filenamens
F602	4C 10 F7	JMP \$F710	ungleich null, dann ok
F605	20 D5 F3	JSR \$F3D5	'missing filename'
F608	20 BF F6	JSR \$F6BF	Filenames auf IEC-Bus
F60B	A5 BA	LDA \$BA	'saving' ausgeben
F60D	20 0C ED	JSR \$ED0C	Geräteadresse
F610	A5 B9	LDA \$B9	LISTEN senden
F612	20 B9 ED	JSR \$EDB9	Sekundär-Adresse
F615	A0 00	LDY #00	für LISTEN senden
F617	20 BE FB	JSR \$FBBE	
F61A	A5 AC	LDA \$AC	Startadresse nach \$AC/\$AD
F61C	20 DD ED	JSR \$EDDD	Startadresse low
F61E	A5 AD	LDA \$AD	senden
F621	20 DD ED	JSR \$EDDD	und high
F624	20 D1 FC	JSR \$FCD1	senden
F627	B0 16	BCS \$F63F	Endadresse schon erreicht?
F629	B1 AC	LDA (\$AC),Y	ja, dann fertig
F62B	20 DD ED	JSR \$EDDD	Programmbytes
F62E	20 E1 FF	JSR \$FFE1	auf IEC-Bus ausgeben
F631	D0 07	BNE \$F63A	STOP-Taste abfragen
F633	20 42 F6	JSR \$F642	nicht gedrückt, dann weitermachen
F636	A9 00	LDA #00	IEC-Bus Kanal schließen
F638	38	SEC	
F639	60	RTS	Flag setzen für 'break' Ausgabe
F63A	20 DB FC	JSR \$FCDB	
F63D	D0 E5	BNE \$F624	laufende Adresse erhöhen
F63F	20 FE ED	JSR \$EDFE	
F642	24 B9	BIT \$B9	UNLISTEN senden
F644	30 11	BMI \$F657	
F646	A5 BA	LDA \$BA	Geräteadresse
F648	20 0C ED	JSR \$ED0C	LISTEN senden
F64B	A5 B9	LDA \$B9	Sekundär-Adresse
F64D	29 EF	AND #\$EF	
F64F	09 E0	ORA #\$E0	
F651	20 B9 ED	JSR \$EDB9	Sekundär-Adresse ausgeben
F654	20 FE ED	JSR \$EDFE	UNLISTEN senden
F657	18	CLC	
F658	60	RTS	
F659	4A	LSR A	Gerätenummer / 2
F65A	B0 03	BCS \$F65F	Band
F65C	4C 13 F7	JMP \$F713	RS 232, 'illegal device number'
F65F	20 D0 F7	JSR \$F7D0	Bandpuffer Startadresse holen



F662	90 BD	BCC \$F5F1	'illegal device number'
F664	20 38 F8	JSR \$F83B	wartet auf Record & Play-Taste
F667	B0 25	BCS \$F68E	STOP, dann Abbruch
F669	20 BF F6	JSR \$F68F	'saving name' ausgeben
F66C	A2 03	LDI \$S03	Header-Typ 3 = Maschinenprogramm (absolut)
F66E	A5 B9	LDA \$B9	Sekundaradresse
F670	29 01	AND \$S01	Bit 0 gesetzt (1 oder 3)
F672	D0 02	BNE \$F676	ja, dann Maschinenprogramm
F674	A2 01	LDI \$S01	Header-Typ 1 = BASIC-Programm (verschieblich)
F676	8A	TXA	
F677	20 6A F7	JSR \$F76A	Header auf Band schreiben
F67A	B0 12	BCS \$F68E	Aussprung bei Stop-Taste
F67C	20 67 F8	JSR \$F867	Programm auf Band schreiben
F67F	B0 0D	BCS \$F68E	Aussprung bei Stop-Taste
F681	A5 B9	LDA \$B9	Sekundaradresse
F683	29 02	AND \$S02	Bit 1 gesetzt (2 oder 3)
F685	F0 06	BED \$F6BD	nein, dann fertig
F687	A9 05	LDA \$S05	EDT Kontrollbyte
F689	20 6A F7	JSR \$F76A	Block auf Band schreiben
F68C	24	.BYTE \$24	
F68D	1B	CLC	
F68E	60	RTS	

*****			'saving' ausgeben
F68F	A5 9D	LDA \$9D	Direkt-Modus ?
F691	10 FB	BPL \$F68E	nein, dann fertig
F693	A0 51	LDY \$S51	Offset für 'saving'
F695	20 2F F1	JSR \$F12F	Meldung ausgeben
F698	4C C1 F5	JMP \$F5C1	Filenamen ausgeben

*****			UDTIM Time erhöhen
F69B	A2 00	LDX \$S00	
F69D	E6 A2	INC \$A2	
F69F	D0 06	BNE \$F6A7	Zeit erhöhen
F6A1	E6 A1	INC \$A1	
F6A3	D0 02	BNE \$F6A7	
F6A5	E6 A0	INC \$A0	
F6A7	3B	SEC	
F6A8	A5 A2	LDA \$A2	
F6AA	E9 01	SBC \$S01	
F6AC	A5 A1	LDA \$A1	Mit Wert für 24 h vergleichen
F6AE	E9 1A	SBC \$S1A	
F6B0	A5 A0	LDA \$A0	
F6B2	E9 4F	SBC \$S4F	
F6B4	90 06	BCC \$F6BC	kleiner, dann ok
F6B6	B6 A0	STX \$A0	
F6B8	B6 A1	STX \$A1	Zeit auf Null setzen
F6BA	B6 A2	STX \$A2	
F6BC	AD 01 DC	LDA \$DC01	
F6BF	CD 01 DC	CMP \$DC01	
F6C2	D0 FB	BNE \$F6BC	
F6C4	AA	TAX	
F6C5	30 13	BNJ \$F6DA	
F6C7	A2 BD	LDX \$SBD	
F6C9	BE 00 DC	STX \$DC00	
F6CC	AE 01 DC	LDX \$DC01	
F6CF	EC 01 DC	CPX \$DC01	Stop-Taste testen
F6D2	D0 FB	BNE \$F6CC	
F6D4	BD 00 DC	STA \$DC00	
F6D7	EB	INX	

F6D8	D0 02	BNE \$F6DC	
F6DA	85 91	STA \$91	Flag für Stop-Taste
F6DC	60	RTS	

*****			TIME holen
F6DD	7B	SEI	
F6DE	A5 A2	LDA \$A2	
F6E0	A6 A1	LDY \$A1	
F6E2	A4 A0	LDY \$A0	

*****			TIME setzen
F6E4	7B	SEI	
F6E5	85 A2	STA \$A2	
F6E7	86 A1	STX \$A1	
F6E9	84 A0	STY \$A0	
F6EB	5B	CLI	
F6EC	60	RTS	

*****			Stop-Taste abfragen
F6ED	A5 91	LDA \$91	Flag
F6EF	C9 7F	CMP #\$7F	auf Code für Stop testen
F6F1	D0 07	BNE \$F6FA	
F6F3	0B	PHP	
F6F4	20 CC FF	JSR \$FFCC	CLRCH
F6F7	85 C6	STA \$C6	Anzahl der gedruckten Tasten
F6F9	2B	PLP	
F6FA	60	RTS	

*****			Meldungen des Betriebssystems ausgeben
F6FB	A9 01	LDA #\$01	
F6FD	2C	.BYTE \$2C	
F6FE	A9 02	LDA #\$02	
F700	2C	.BYTE \$2C	
F701	A9 03	LDA #\$03	
F703	2C	.BYTE \$2C	
F704	A9 04	LDA #\$04	
F706	2C	.BYTE \$2C	
F707	A9 05	LDA #\$05	
F709	2C	.BYTE \$2C	
F70A	A9 06	LDA #\$06	
F70C	2C	.BYTE \$2C	
F70D	A9 07	LDA #\$07	
F70F	2C	.BYTE \$2C	
F710	A9 08	LDA #\$08	
F712	2C	.BYTE \$2C	
F713	A9 09	LDA #\$09	
F715	4B	PHA	Fehlernummer merken
F716	20 CC FF	JSR \$FFCC	CLRCH
F719	A0 00	LDY #\$00	
F71B	24 9D	BIT \$9D	Flag für Error-Ausgabe testen
F71D	50 0A	BVC \$F729	nicht gesetzt, dann übergehen
F71F	20 2F F1	JSR \$F12F	'I/O ERROR #' ausgeben
F722	6B	PLA	
F723	4B	PHA	Fehlernummer holen
F724	09 30	DRA #\$30	nach ASCII
F726	20 D2 FF	JSR \$FFD2	und ausgeben
F729	6B	PLA	
F72A	3B	SEC	
F72B	60	RTS	

```

***** Program Header vom Band lesen
F72C A5 93 LDA $93 Load/Verify Flag retten
F72E 4B PHA
F72F 20 41 F8 JSR $F841 Block vom Band lesen
F732 68 PLA
F733 05 93 STA $93 Load/Verify Flag zurueckholen
F735 B0 32 BCS $F769 Fehler, dann beenden
F737 A0 00 LDY #00
F739 B1 02 LDA ($B2),Y Header-Typ testen
F73B C9 05 CMP #05 EOT ?
F73D F0 2A BEQ $F769
F73F C9 01 CMP #01 BASIC-Programm ?
F741 F0 08 BEQ $F74B
F743 C9 03 CMP #03 Maschinen-Programm ?
F745 F0 04 BEQ $F74B
F747 C9 04 CMP #04 Daten-Header ?
F749 D0 E1 BNE $F72C nein
F74B AA TAX
F74C 24 9D BIT $9D Direkt-Modus ?
F74E 10 17 BPL $F767 nein, dann weiter
F750 A0 63 LDY #63
F752 20 2F F1 JSR $F12F 'found' ausgeben
F755 A0 05 LDY #05 Offset des Filenamens
F757 B1 02 LDA ($B2),Y
F759 20 D2 FF JSR $FFD2 Filenamens ausgeben
F75C C8 INY
F75D C0 15 CPY #15
F75F D0 F6 BNE $F757
F761 A5 A1 LDA $A1 Akku mit mittelwertigem Time-Byte laden
F763 20 E0 E4 JSR $E4E0 wartet auf Commodore-Taste oder Zeitschleife
F766 EA NOP
F767 18 CLC bei Fehler ist C=1
F768 88 DEY
F769 60 RTS

```

```

***** Header generieren und auf Band schreiben
F76A 05 9E STA $9E Header-Typ
F76C 20 D0 F7 JSR $F7D0 Bandpufferadresse holen
F76F 90 5E BCC $F7CF
F771 A5 C2 LDA $C2
F773 48 PHA Startadresse
F774 A5 C1 LDA $C1
F776 48 PHA und
F777 A5 AF LDA $AF
F779 48 PHA Endadresse merken
F77A A5 AE LDA $AE
F77C 48 PHA
F77D A0 BF LDY #BF Pufferlange - 1
F77F A9 20 LDA #20
F781 91 B2 STA ($B2),Y Bandpuffer loschen
F783 88 DEY
F784 D0 FB BNE $F781
F786 A5 9E LDA $9E
F788 91 B2 STA ($B2),Y Header-Typ
F78A C8 INY
F78B A5 C1 LDA $C1
F78D 91 B2 STA ($B2),Y Startadresse
F78F C8 INY
F790 A5 C2 LDA $C2
F792 91 B2 STA ($B2),Y

```

F794	C8	INY	
F795	A5 AE	LDA \$AE	
F797	91 B2	STA (\$B2),Y	Endadresse
F799	C8	INY	
F79A	A5 AF	LDA \$AF	
F79C	91 B2	STA (\$B2),Y	
F79E	C8	INY	
F79F	84 9F	STY \$9F	
F7A1	A0 00	LDY #000	
F7A3	84 9E	STY \$9E	Zähler für Filenamenlänge
F7A5	A4 9E	LDY \$9E	
F7A7	C4 B7	CPY \$B7	mit Länge vergleichen
F7A9	F0 0C	BEQ \$F7B7	alle Buchstaben, dann weiter
F7AB	B1 BB	LDA (\$BB),Y	Filenamen holen
F7AD	A4 9F	LDY \$9F	
F7AF	91 B2	STA (\$B2),Y	in Header schreiben
F7B1	E6 9E	INC \$9E	
F7B3	E6 9F	INC \$9F	
F7B5	D0 EE	BNE \$F7A5	
F7B7	20 D7 F7	JSR \$F7D7	Start- und Endadresse auf Bandpuffer
F7BA	A9 69	LDA #69	
F7BC	B5 AB	STA \$AB	Checksumme für Header bzw Datenblock = \$69
F7BE	20 6B F8	JSR \$F86B	Block auf Band schreiben
F7C1	AB	TAY	
F7C2	6B	PLA	
F7C3	85 AE	STA \$AE	
F7C5	6B	PLA	End-
F7C6	B5 AF	STA \$AF	
F7C8	6B	PLA	und
F7C9	B5 C1	STA \$C1	
F7CB	6B	PLA	Startadresse zurückholen
F7CC	B5 C2	STA \$C2	
F7CE	9B	TYA	
F7CF	60	RTS	
*****			Bandpuffer Startadresse holen
F7D0	A6 B2	LDX \$B2	
F7D2	A4 B3	LDY \$B3	
F7D4	C0 02	CPY #02	Adresse kleiner \$200 ?
F7D6	60	RTS	
*****			
F7D7	20 D0 F7	JSR \$F7D0	Bandpuffer-Adresse holen
F7DA	BA	TXA	
F7DB	85 C1	STA \$C1	Startadresse = Start Bandpuffer
F7DD	18	CLC	
F7DE	69 C0	ADC #\$C0	
F7E0	B5 AE	STA \$AE	
F7E2	9B	TYA	Endadresse = Startadresse + Länge (192)
F7E3	85 C2	STA \$C2	
F7E5	69 00	ADC #000	
F7E7	B5 AF	STA \$AF	
F7E9	60	RTS	
*****			Bandheader nach Namen suchen
F7EA	20 2C F7	JSR \$F72C	nächsten Bandheader suchen
F7ED	B0 10	BCS \$F80C	EOT, dann fertig
F7EF	A0 05	LDY #05	Offset für Filenamen im Header
F7F1	84 9F	STY \$9F	
F7F3	A0 00	LDY #000	

F7F5	B4 9E	STY \$9E	Zähler für Länge des Filenamens
F7F7	C4 B7	CPY \$B7	mit Länge des gesuchten Namens vergleichen
F7F9	F0 10	BEQ \$F80B	gleich, dann gefunden
F7FB	B1 B6	LDA (\$B8),Y	Buchstaben des Filenamens
F7FD	A4 9F	LDY \$9F	
F7FF	D1 52	CMP (\$B2),Y	mit Filenamem im Header vergleichen
F801	D0 E7	BNE \$F7EA	ungleich, dann nächsten Header testen
F803	E6 9E	INC \$9E	
F805	E6 9F	INC \$9F	Zähler erhöhen
F807	A4 9E	LDY \$9E	
F809	D0 EC	BNE \$F7F7	weitere Buchstaben vergleichen
F80B	1B	CLC	
F80C	60	RTS	

*****			Bandpufferzeiger erhöhen
F80D	20 D0 F7	JSR \$F7D0	Bandpufferadresse holen
F810	E6 A6	INC \$A6	Zeiger erhöhen
F812	A4 A6	LDY \$A6	
F814	C0 C0	CPY \$C0	mit Maximalwert (192) vergleichen
F816	60	RTS	

*****			Wartet auf Band-Taste
F817	20 2E FB	JSR \$F82E	fragt Band-Taste ab
F81A	F0 1A	BEQ \$F836	gedrückt, dann fertig
F81C	A0 1B	LDY \$1B	Offset für 'press play on tape'
F81E	20 2F F1	JSR \$F12F	ausgeben
F821	20 D0 FB	JSR \$F8D0	testet auf STOP-Taste
F824	20 2E FB	JSR \$F82E	fragt Band-Taste ab
F827	D0 FB	BNE \$F821	
F829	A0 6A	LDY \$6A	Offset für 'ok'
F82B	4C 2F F1	JMP \$F12F	ausgeben
*****			Abfrage ob Band-Taste gedrückt
F82E	A9 10	LDA \$10	Bit 4 testen
F830	24 01	BIT \$01	
F832	D0 02	BNE \$F836	
F834	24 01	BIT \$01	
F836	1B	CLC	ja, dann Z=1, sonst Z=0
F837	60	RTS	

*****			Wartet auf Bandtaste für Schreiben
F838	20 2E FB	JSR \$F82E	fragt Bandtaste ab
F83B	F0 F9	BEQ \$F836	gedrückt, dann fertig
F83D	A0 2E	LDY \$2E	Offset für 'press record & play on tape'
F83F	D0 DD	BNE \$F81E	weiter wie oben

*****			Block vom Band lesen
F841	A9 00	LDA \$00	
F843	B5 90	STA \$90	Status und
F845	B5 93	STA \$93	Verify-Flag löschen
F847	20 D7 F7	JSR \$F7D7	Bandpufferadresse holen

*****			Programm vom Band laden
F84A	20 17 FB	JSR \$F817	wartet auf Play-Taste
F84D	B0 1F	BCS \$F86E	STOP-Taste gedrückt ?
F84F	78	SEI	
F850	A9 00	LDA \$00	
F852	B5 AA	STA \$AA	
F854	B5 B4	STA \$B4	
F856	B5 B0	STA \$B0	Arbeitsspeicher für IRQ-Routine löschen
F858	B5 9E	STA \$9E	

F85A	85 9F	STA \$9F	
F85C	85 9C	STA \$9C	
F85E	A9 90	LDA \$90	IRQ an Pin 'Flag'
F860	A2 0E	LDX \$0E	Nummer des IRQ-Vektors, \$F92C
F862	D0 11	BNE \$F875	
*****			Bandpuffer auf Band schreiben
F864	20 D7 F7	JSR \$F7D7	Bandpufferadresse holen
F867	A9 14	LDA \$814	Checksumme
F869	85 AB	STA \$AB	für Datenblock bzw. Header = \$14
*****			Block bzw. Programm auf Band schreiben
F86B	20 38 F8	JSR \$F838	wartet auf Record & Play Taste
F86E	B0 6C	BCS \$F8DC	STOP-Taste gedrückt ?
F870	78	SEI	
F871	A9 82	LDA \$82	IRQ bei Unterlauf von Timer B
F873	A2 08	LDY \$08	Nummer des IRQ-Vektors, \$FC6A
F875	A0 7F	LDY \$7F	
F877	9C 0D DC	STY \$DC0D	IRQ-Maske löschen
F87A	8D 0D DC	STA \$DC0D	und neu setzen
F87D	AD 0E DC	LDA \$DC0E	CRA
F880	09 19	DRA \$19	
F882	9D 0F DC	STA \$DC0F	CRA, IRQ an Timer B
F885	29 91	AND \$91	
F887	8D A2 02	STA \$02A2	Timer A Kontroll-Flag
F88A	20 A4 F0	JSR \$F0A4	
F88D	AD 11 D0	LDA \$D011	
F890	29 EF	AND \$EF	Bildschirm dunkel tasten
F892	8D 11 D0	STA \$D011	
F895	AD 14 03	LDA \$0314	IRQ-Vector
F898	8D 9F 02	STA \$029F	
F89B	AD 15 03	LDA \$0315	nach \$29F/\$2A0 speichern
F89E	8D A0 02	STA \$02A0	
F8A1	20 8D FC	JSR \$FC8D	IRQ-Vector für Band I/O setzen (X-indiziert)
F8A4	A9 02	LDA \$02	
F8A6	85 BE	STA \$BE	Anzahl der Blocks zu lesen
F8AB	20 97 FB	JSR \$FB97	serielle Ausgabe vorbereiten Bit-Zähler setzen
F8AB	A5 01	LDA \$01	
F8AD	29 1F	AND \$1F	Band-Motor einschalten
F8AF	85 01	STA \$01	
F8B1	85 C0	STA \$C0	Flag für Band-Motor setzen
F8B3	A2 FF	LDX \$FF	
F8B5	A0 FF	LDY \$FF	
F8B7	88	DEY	
F8B8	D0 FD	BNE \$F8B7	Verzögerungsschleife für Bandhochlaufzeit
F8BA	CA	DEX	
F8BB	D0 F8	BNE \$F8B5	
F8BD	58	CLI	Interrupt für Band I/O freigeben
*****			I/O Abschluß abwarten
F8BE	AD A0 02	LDA \$02A0	IRQ-Vector wieder auf Standard-Wert ?
F8C1	CD 15 03	CMP \$0315	
F8C4	18	CLC	
F8C5	F0 15	BEQ \$F8DC	ja, dann fertig
F8C7	20 D0 F8	JSR \$F8D0	Test auf Stop-Taste
F8CA	20 BC F6	JSR \$F6BC	bei gedrückter Stop-Taste Flag setzen
F8CD	4C BE F8	JMP \$F8BE	weiter warten
*****			testet auf Stop-Taste
F8D0	20 E1 FF	JSR \$FFE1	Stop-Taste abfragen

F8D3	18	CLC	
F8D4	D0 08	BNE \$F8E1	nein, dann Rückkehr
F8D6	20 93 FC	JSR \$FC93	Band-Motor aus, normalen IRQ wiederherstellen
F8D9	38	SEC	Kennzeichen für Abbruch
F8DA	68	PLA	
F8DB	68	PLA	Rucksprung Adresse löschen
F8DC	A9 00	LDA #\$00	
F8DE	8D A0 02	STA \$02A0	Kennzeichen für normalen IRQ
F8E1	60	RTS	

\*\*\*\*\* Band für Lesen vorbereiten

F8E2	86 B1	STX \$B1	
F8E4	A5 B0	LDA \$B0	
F8E6	0A	ASL A	
F8E7	0A	ASL A	
F8E8	18	CLC	
F8E9	65 B0	ADC \$B0	
F8EB	18	CLC	
F8EC	65 B1	ADC \$B1	
F8EE	85 B1	STA \$B1	
F8F0	A9 00	LDA #\$00	
F8F2	24 B0	BIT \$B0	
F8F4	30 01	BMI \$F8F7	
F8F6	2A	ROL A	
F8F7	06 B1	ASL \$B1	
F8F9	2A	ROL A	
F8FA	06 B1	ASL \$B1	
F8FC	2A	ROL A	
F8FD	AA	TAX	
F8FE	AD 06 DC	LDA \$DC06	Timer B Lo
F901	C9 16	CMP #\$16	
F903	90 F9	BCC \$F8FE	
F905	65 B1	ADC \$B1	
F907	8D 04 DC	STA \$DC04	Timer A Lo
F90A	8A	TxA	
F90B	6D 07 DC	ADC \$DC07	Timer B Hi
F90E	8D 05 DC	STA \$DC05	Timer A Hi
F911	AD A2 02	LDA \$02A2	
F914	8D 0E DC	STA \$DC0E	
F917	8D A4 02	STA \$02A4	
F91A	AD 0D DC	LDA \$DC0D	Input vom Band lesen, Pin 'Flag'
F91D	29 10	AND #\$10	Bit isolieren
F91F	F0 09	BEQ \$F92A	
F921	A9 F9	LDA #\$F9	
F923	48	PHA	Rücksprungadresse auf Stack
F924	A9 2A	LDA #\$2A	
F926	48	PHA	
F927	4C 43 FF	JMP \$FF43	zum Interrupt
F92A	58	CLI	
F92B	60	RTS	

\*\*\*\*\* Interrupt-Routine für Band lesen

F92C	AE 07 DC	LDI \$DC07	Timer B Hi
F92F	A0 FF	LDY \$FFF	
F931	98	TYA	
F932	ED 06 DC	SBC \$DC06	Timer B Lo
F935	EC 07 DC	CPX \$DC07	Timer B Hi
F938	D0 F2	BNE \$F92C	
F93A	86 B1	STX \$B1	
F93C	AA	TAX	

F93D	BC 06 DC	STY \$DC06
F940	BC 07 DC	STY \$DC07
F943	AY 19	LDA #19
F945	BD 0F DC	STA \$DC0F
F948	AD 00 DC	LDA \$DC0D
F94B	BD A3 02	STA \$02A3
F94E	9B	TYA
F94F	E5 B1	SBC \$B1
F951	B6 B1	STX \$B1
F953	4A	LSR A
F954	66 B1	ROR \$B1
F956	4A	LSR A
F957	66 B1	ROR \$B1
F959	A5 B0	LDA \$B0
F95B	1B	CLC
F95C	69 3C	ADC #3C
F95E	C5 B1	CMP \$B1
F960	B0 4A	BCS \$F9AC
F962	A6 9C	LDX \$9C
F964	F0 03	BEQ \$F969
F966	4C 60 FA	JMP \$FA60

Timer B Lo

Timer B Hi

IRQ von Timer B

Input vom Band, Pin 'Flag'

F969	A6 A3	LDX \$A3
F96B	30 1B	BMI \$F9B9
F96D	A2 00	LDX #00
F96F	69 30	ADC #30
F971	65 B0	ADC \$B0
F973	C5 B1	CMP \$B1
F975	B0 1C	BCS \$F993
F977	EB	INX
F97B	69 26	ADC #26
F97A	65 B0	ADC \$B0
F97C	C5 B1	CMP \$B1
F97E	B0 17	BCS \$F997
F980	69 2C	ADC #2C
F982	65 B0	ADC \$B0
F984	C5 B1	CMP \$B1
F986	90 03	BCC \$F98B
F988	4C 10 FA	JMP \$FA10
F98B	A5 B4	LDA \$B4
F98D	F0 1D	BEQ \$F9AC
F98F	B5 AB	STA \$AB
F991	D0 19	BNE \$F9AC
F993	E6 A9	INC \$A9
F995	B0 02	BCS \$F999
F997	C6 A9	DEC \$A9
F999	3B	SEC
F99A	E9 13	SBC #13
F99C	E5 B1	SBC \$B1
F99E	65 92	ADC \$92
F9A0	B5 92	STA \$92
F9A2	A5 A4	LDA \$A4
F9A4	49 01	EDR #01
F9A6	B5 A4	STA \$A4
F9A8	F0 2B	BEQ \$F9D5
F9AA	B6 D7	STX \$D7
F9AC	A5 B4	LDA \$B4
F9AE	F0 22	BEQ \$F9D2
F9B0	AD A3 02	LDA \$02A3
F9B3	29 01	AND #01



F9B5	D0 05	BNE \$F9BC	
F9B7	AD A4 02	LDA \$02A4	
F9BA	D0 16	BNE \$F9D2	
F9BC	A9 00	LDA #00	
F9BE	85 A4	STA \$A4	
F9C0	BD A4 02	STA \$02A4	
F9C3	A5 A3	LDA \$A3	
F9C5	10 30	BPL \$F9F7	
F9C7	30 BF	BMI \$F98B	
F9C9	A2 A6	LDX #A6	
F9CB	20 E2 FB	JSR \$F8E2	
F9CE	A5 9B	LDA \$9B	
F9D0	D0 B9	BNE \$F98B	
F9D2	4C BC FE	JMP \$F8BC	Rückkehr vom Interrupt

F9D5	A5 92	LDA \$92	
F9D7	F0 07	BEQ \$F9E0	
F9D9	30 03	BMI \$F9DE	
F9DB	C6 B0	DEC #B0	
F9DD	2C	.BYTE \$2C	
F9DE	E6 B0	INC #B0	
F9E0	A9 00	LDA #00	
F9E2	85 92	STA \$92	
F9E4	E4 D7	CPY \$D7	
F9E6	D0 0F	BNE \$F9F7	
F9E8	BA	TXA	
F9E9	D0 A0	BNE \$F98B	
F9EB	A5 A9	LDA \$A9	
F9ED	30 B0	BMI \$F9AC	
F9EF	C9 10	CMP #10	
F9F1	90 B9	BCC \$F9AC	
F9F3	B5 96	STA \$96	
F9F5	B0 B5	BCE \$F9AC	
F9F7	8A	TXA	
F9F8	45 9B	EOR #9B	
F9FA	85 9B	STA \$9B	
F9FC	A5 B4	LDA \$B4	
F9FE	F0 D2	BEQ \$F9D2	
FA00	C6 A3	DEC \$A3	
FA02	30 C5	BMI \$F9C9	
FA04	46 D7	LSR \$D7	
FA06	66 BF	ROR \$BF	
FA08	A2 DA	LDX #DA	
FA0A	20 E2 FB	JSR \$F8E2	
FA0D	4C BC FE	JMP \$F8BC	Rückkehr vom Interrupt
FA10	A5 96	LDA \$96	
FA12	F0 04	BEQ \$FA18	
FA14	A5 B4	LDA \$B4	
FA16	F0 07	BEQ \$FA1F	
FA18	A5 A3	LDA \$A3	
FA1A	30 03	BMI \$FA1F	
FA1C	4C 97 F9	JMP \$F997	
FA1F	46 B1	LSR #B1	
FA21	A9 93	LDA #93	
FA23	30	SEC	
FA24	E5 B1	SBC #B1	
FA26	65 B0	ADC #B0	
FA28	0A	ASL A	
FA29	AA	TAX	
FA2A	20 E2 FB	JSR \$F8E2	

FA2D	E6 9C	INC \$9C	
FA2F	A5 B4	LDA \$B4	
FA31	D0 11	BNE \$FA44	
FA33	A5 96	LDA \$96	
FA35	F0 26	BEQ \$FA5D	
FA37	85 A8	STA \$A8	
FA39	A9 00	LDA #\$00	
FA3B	85 96	STA \$96	
FA3D	A9 B1	LDA \$B1	
FA3F	8D 0D DC	STA \$DC0D	IRQ bei Unterlauf Timer A
FA42	85 B4	STA \$B4	
FA44	A5 96	LDA \$96	
FA46	85 B5	STA \$B5	
FA48	F0 09	BEQ \$FA53	
FA4A	A9 00	LDA #\$00	
FA4C	85 B4	STA \$B4	
FA4E	A9 01	LDA \$01	
FA50	8D 0D DC	STA \$DC0D	IRQ-Flag wieder löschen
FA53	A5 BF	LDA \$BF	
FA55	85 BD	STA \$BD	
FA57	A5 A8	LDA \$A8	
FA59	05 A9	ORA \$A9	
FA5B	85 B6	STA \$B6	
FA5D	4C 9C FE	JMP \$FE9C	Rückkehr vom Interrupt
FA60	20 97 FB	JSR \$FB97	Bitzähler für serielle Ausgabe setzen
FA63	B5 9C	STA \$9C	
FA65	A2 DA	LDX \$DA	
FA67	20 E2 FB	JSR \$FBE2	
FA6A	A5 BE	LDA \$BE	
FA6C	F0 02	BEN \$FA70	
FA6E	85 A7	STA \$A7	
FA70	A9 0F	LDA \$0F	
FA72	24 AA	BIT \$AA	
FA74	10 17	BPL \$FABD	
FA76	A5 B5	LDA \$B5	
FA78	D0 0C	BNE \$FAB6	
FA7A	A6 BE	LDX \$BE	
FA7C	CA	DEX	
FA7D	D0 0B	BNE \$FABA	
FA7F	A9 0B	LDA \$0B	'long block' error
FAB1	20 1C FE	JSR \$FE1C	Status setzen
FAB4	D0 04	BNE \$FABA	
FAB6	A9 00	LDA \$00	
FAB8	85 AA	STA \$AA	
FABA	4C BC FE	JMP \$FEBC	Rückkehr vom Interrupt
FABD	70 31	BVS \$FAC0	
FABF	D0 18	BNE \$FAA9	
FA91	A5 B5	LDA \$B5	
FA93	D0 F5	BNE \$FABA	
FA95	A5 B6	LDA \$B6	
FA97	D0 F1	BNE \$FABA	
FA99	A5 A7	LDA \$A7	
FA9B	4A	LSR A	
FA9C	A5 BD	LDA \$BD	
FA9E	30 03	BMI \$FAA3	
FAA0	90 18	BCC \$FABA	
FAA2	18	CLC	
FAA3	B0 15	BCS \$FABA	
FAA5	29 0F	AND #\$0F	
FAA7	B5 AA	STA \$AA	

FAA9	C6 AA	DEC \$AA	
FAAB	D0 DD	BNE \$FABA	
FAAD	A9 40	LDA #40	
FAAF	85 AA	STA \$AA	
FAB1	20 BE FB	JSR \$FB8E	
FAB4	A9 00	LDA #00	
FAB6	B5 AB	STA \$AB	
FAB8	F0 D0	BEQ \$FABA	
FABA	A9 80	LDA #80	
FABC	85 AA	STA \$AA	
FABE	D0 CA	BNE \$FABA	
FAC0	A5 B5	LDA \$B5	
FAC2	F0 0A	BEQ \$FACE	
FAC4	A9 04	LDA #04	'short block' error
FAC6	20 1C FE	JSR \$FE1C	Status setzen
FAC9	A9 00	LDA #00	
FACB	4C 4A FB	JMP \$FB4A	
FACE	20 D1 FC	JSR \$FCD1	Endadresse schon erreicht ?
FAD1	90 03	BCC \$FAD6	nein
FAD3	4C 48 FB	JMP \$FB48	
FAD6	A6 A7	LDX \$A7	
FADB	CA	DEX	
FAD9	F0 2D	BEQ \$FB08	
FADB	A5 93	LDA \$93	Load/Verify-Flag
FADD	F0 0C	BEQ \$FAEB	Load ?
FADF	A0 00	LDY #00	
FAE1	A5 B0	LDA \$BD	gelesenes Byte
FAE3	D1 AC	CMP (\$AC),Y	vergleichen
FAE5	F0 04	BEQ \$FAEB	ubereinstimmung ?
FAE7	A9 01	LDA #01	
FAE9	B5 B6	STA \$B6	Flag setzen
FAEB	A5 B6	LDA \$B6	
FAED	F0 4B	BEQ \$FB3A	Byte in Ordnung ?
FAEF	A2 3D	LDX #3D	
FAF1	E4 9E	CPX \$9E	
FAF3	90 3E	BCC \$FB33	
FAF5	A6 9E	LDX \$9E	
FAF7	A5 AD	LDA \$AD	
FAF9	9D 01 01	STA \$0101,X	
FAFC	A5 AC	LDA \$AC	Daten für second pass Korrektur
FAFE	9D 00 01	STA \$0100,X	
FB01	EB	INX	
FB02	EB	INX	
FB03	B6 9E	STX \$9E	
FB05	4C 3A FB	JMP \$FB3A	
FB08	A6 9F	LDX \$9F	
FB0A	E4 9E	CPX \$9E	
FB0C	F0 35	BEQ \$FB43	
FB0E	A5 AC	LDA \$AC	
FB10	D0 00 01	CMP \$0100,X	Fehlerkorrektur bei Pass 2
FB13	D0 2E	BNE \$FB43	
FB15	A5 AD	LDA \$AD	
FB17	D0 01 01	CMP \$0101,X	
FB1A	D0 27	BNE \$FB43	
FB1C	E6 9F	INC \$9F	
FB1E	E6 9F	INC \$9F	
FB20	A5 93	LDA \$93	
FB22	F0 08	BEQ \$FB2F	
FB24	A5 B0	LDA \$BD	gelesenes Byte
FB26	A0 00	LDY #00	

FB2B	D1 AC	CMP (\$AC),Y	mit Speicherinhalt vergleichen
FB2A	F0 17	BEQ \$FB43	
FB2C	C8	INY	
FB2D	84 B6	STY \$B6	
FB2F	A5 B6	LDA \$B6	
FB31	F0 07	BEQ \$FB3A	
FB33	A9 10	LDA \$B10	'second pass' error
FB35	20 1C FE	JSR \$FE1C	Status setzen
FB38	D0 09	BNE \$FB43	
FB3A	A5 93	LDA \$93	Verify ?
FB3C	D0 05	BNE \$FB43	'a
FB3E	AB	TAY	
FB3F	A5 BD	LDA \$BD	gelesenes Byte
FB41	91 AC	STA (\$AC),Y	speichern
FB43	20 DB FC	JSR \$FCDB	Adresszeiger erhöhen
FB46	D0 43	BNE \$FB8B	Rückkehr vom Interrupt
FB48	A9 B0	LDA \$B0	
FB4A	B5 AA	STA \$AA	
FB4C	7B	SEI	
FB4D	A2 01	LDX \$A01	
FB4F	BE 0D DC	STX \$DC0D	IRQ vom Timer A verhindern
FB52	AE 0D DC	LDX \$DC0D	IRQ-Flag löschen
FB55	A6 BE	LDX \$BE	Pass-Zähler
FB57	CA	DEX	erniedrigen
FB58	30 02	BMI \$FB5C	
FB5A	B6 BE	STX \$BE	
FB5C	C6 A7	DEC \$A7	
FB5E	F0 0B	BEQ \$FB6B	
FB60	A5 9E	LDA \$9E	
FB62	D0 27	BNE \$FB8B	Rückkehr vom Interrupt
FB64	B5 BE	STA \$BE	
FB66	F0 23	BEQ \$FB8B	Rückkehr vom Interrupt
FB68	20 93 FC	JSR \$FC93	ein Pass beendet
FB6B	20 BE FB	JSR \$FB8E	Adresse wieder auf Programmstart
FB6E	A0 00	LDY \$A00	
FB70	84 AB	STY \$AB	
FB72	B1 AC	LDA (\$AC),Y	Program Checksumme berechnen
FB74	45 AB	EOR \$AB	
FB76	B5 AB	STA \$AB	
FB78	20 DB FC	JSR \$FCDB	Adresszeiger erhöhen
FB7B	20 D1 FC	JSR \$FCD1	Endadresse schon erreicht ?
FB7E	90 F2	BCC \$FB72	nein, weiter vergleichen
FB80	A5 AB	LDA \$AB	berechnete Checksumme
FB82	45 BD	EOR \$BD	mit Checksumme vom Band vergleichen
FB84	F0 05	BEQ \$FB8B	Checksumme ok ?
FB86	A9 20	LDA \$A20	'checksum' error
FB88	20 1C FE	JSR \$FE1C	Status setzen
FB8B	4C BC FE	JMP \$FEBF	Rückkehr vom Interrupt
FB8E	A5 C2	LDA \$C2	
FB90	B5 AD	STA \$AD	
FB92	A5 C1	LDA \$C1	
FB94	B5 AC	STA \$AC	
FB96	60	RTS	
#####			Bitzähler für serielle Ausgabe setzen
FB97	A9 0B	LDA \$A0B	
FB99	B5 A3	STA \$A3	8 Bits
FB9B	A9 00	LDA \$A00	
FB9D	B5 A4	STA \$A4	
FB9F	B5 AB	STA \$AB	

FBA1	85 9B	STA \$9B
FBA3	85 A9	STA \$A9
FBA5	60	RTS

\*\*\*\*\*

FBA6	A5 BD	LDA \$BD
FBA8	4A	LSR A
FBA9	A9 60	LDA \$60
FBA8	90 02	BCC \$FBAF
FBA8	A9 80	LDA \$80
FBAF	A2 00	LDX \$00
FBB1	8D 06 DC	STA \$DC06
FBB4	8E 07 DC	STX \$DC07
FBB7	AD 0D DC	LDA \$DC0D
FBB8	A9 19	LDA \$19
FBB8	8D 0F DC	STA \$DC0F
FBBF	A5 01	LDA \$01
FBC1	49 08	EOR \$08
FBC3	85 01	STA \$01
FBC5	29 08	AND \$08
FBC7	60	RTS

Ein Bit auf Band schreiben  
 Bit in \$BD  
 Bit 0 in Carry  
 Zeit für '1' Bit  
 Zeit für '0' Bit  
 Timer B low  
 Timer B high  
 Interrupt-Flag löschen  
 Timer starten  
 Ausgabe-Bit für Band invertieren

FBC8	38	SEC
FBC9	66 B6	ROR \$B6
FBC8	30 3C	BMI \$FC09

Rückkehr vom Interrupt  
 Interrupt-Routine für Band schreiben

\*\*\*\*\*

FBCD	A5 A8	LDA \$A8
FBCF	D0 12	BNE \$FRE3
FBD1	A9 10	LDA \$10
FBD3	A2 01	LDX \$01
FBD5	20 B1 FB	JSR \$FBB1
FBD8	D0 2F	BNE \$FC09
FBD8	E6 A8	INC \$A8
FBD8	A5 B6	LDA \$B6
FBD8	10 29	BPL \$FC09
FBE0	4C 57 FC	JMP \$FC57
FBE3	A5 A9	LDA \$A9
FBE5	D0 09	BNE \$FBF0
FBE7	20 A1 FB	JSR \$FBAD
FBEA	D0 1D	BNE \$FC09
FBE8	E6 A9	INC \$A9
FBE8	D0 19	BNE \$FC09
FBF0	20 A6 FB	JSR \$FBA6
FBF3	D0 14	BNE \$FC09
FBF5	A5 A4	LDA \$A4
FBF7	49 01	EOR \$01
FBF9	85 A4	STA \$A4
FBFB	F0 0F	BEQ \$FC0C
FBFD	A5 BD	LDA \$BD
FBFF	49 01	EOR \$01
FC01	85 BD	STA \$BD
FC03	29 01	AND \$01
FC05	45 9B	EOR \$9B
FC07	85 9B	STA \$9B
FC09	4C BC FE	JMP \$FEB8
FC0C	46 BD	LSR \$BD
FC0E	C6 A3	DEC \$A3
FC10	A5 A3	LDA \$A3
FC12	F0 3A	BEQ \$FC4E
FC14	10 F3	BPL \$FC09

Takt auf Band schreiben  
 Rückkehr vom Interrupt  
 Rückkehr vom Interrupt  
 zweiten Block schreiben  
 '0' Bit schreiben  
 Rückkehr vom Interrupt  
 Rückkehr vom Interrupt  
 Bit auf Band schreiben  
 Rückkehr vom Interrupt

Bit für Ausgabe invertieren

Rückkehr vom Interrupt  
 nächstes Bit in Position 0  
 Bitzähler erniedrigen  
 nächstes Bit ausgeben  
 Rückkehr vom Interrupt

FC16	20 97 FB	JSR \$FB97	Bitzähler wieder auf 8 setzen
FC19	58	CLI	
FC1A	A5 A5	LDA \$A5	
FC1C	F0 12	BEQ \$FC30	
FC1E	A2 00	LDX ##00	
FC20	B6 D7	STX \$D7	
FC22	C6 A5	DEC \$A5	
FC24	A6 BE	LDX \$BE	
FC26	E0 02	CPX ##02	
FC28	D0 02	BNE \$FC2C	
FC2A	09 80	ORA ##80	
FC2C	B5 BD	STA \$BD	
FC2E	D0 D9	BNE \$FC09	Rückkehr vom Interrupt
FC30	20 D1 FC	JSR \$FCD1	Endadresse schon erreicht ?
FC33	90 0A	BCC \$FC3F	
FC35	D0 91	BNE \$FBC9	
FC37	E6 AD	INC \$AD	
FC39	A5 D7	LDA \$D7	
FC3B	B5 BD	STA \$BD	
FC3D	B0 CA	BCS \$FC09	Rückkehr vom Interrupt
FC3F	A0 00	LDY ##00	
FC41	B1 AC	LDA (\$AC),Y	zu speicherndes Byte
FC43	B5 BD	STA \$BD	
FC45	45 D7	EOR \$D7	
FC47	B5 D7	STA \$D7	
FC49	20 DB FC	JSR \$FCDB	Adresszeiger erhöhen
FC4C	D0 BB	BNE \$FC09	Rückkehr vom Interrupt
FC4E	A5 9B	LDA \$9B	
FC50	49 01	EOR ##01	
FC52	B5 BD	STA \$BD	
FC54	4C BC FE	JMP \$FEBF	Rückkehr vom Interrupt
FC57	C6 BE	DEC \$BE	Zähler für Blocks erniedrigen
FC59	D0 03	BNE \$FC5E	noch ein Block ?
FC5B	20 CA FC	JSR \$FCCA	nein, Band-Motor aus
FC5E	A9 50	LDA ##50	
FC60	B5 A7	STA \$A7	
FC62	A2 08	LDX ##08	
FC64	78	SEI	
FC65	20 BD FC	JSR \$FCBD	IRQ auf \$FC6A
FC68	D0 EA	BNE \$FC54	Rückkehr vom Interrupt
*****			Interrupt-Routine für Band schreiben
FC6A	A9 78	LDA ##78	
FC6C	20 AF FB	JSR \$FBAF	Bit auf Band schreiben
FC6F	D0 E3	BNE \$FC54	Rückkehr vom Interrupt
FC71	C6 A7	DEC \$A7	
FC73	D0 DF	BNE \$FC54	Rückkehr vom Interrupt
FC75	20 97 FB	JSR \$FB97	Bitzähler für serielle Ausgabe setzen
FC78	C6 AB	DEC \$AB	
FC7A	10 D8	BPL \$FC54	Rückkehr vom Interrupt
FC7C	A2 0A	LDX ##0A	
FC7E	20 BD FC	JSR \$FCBD	IRQ auf \$FBFD
FC81	58	CLI	
FC82	E6 AB	INC \$AB	
FC84	A5 BE	LDA \$BE	
FC86	F0 30	BEQ \$FCB9	
FC88	20 BE FB	JSR \$FBBE	Adresse wieder auf Anfang setzen
FC8B	A2 09	LDX ##09	
FC8D	B6 A5	STX \$A5	
FC8F	B6 B6	STX \$B6	
FC91	D0 B3	BNE \$FC16	

FC93	08	PHP	
FC94	78	SEI	
FC95	AD 11 D0	LDA \$D011	
FC98	09 10	ORA #\$10	Bildschirm wieder einschalten
FC9A	8D 11 D0	STA \$D011	
FC9D	20 CA FC	JSR \$FCCA	Rekordermotor ausschalten
FCA0	A9 7F	LDA #\$7F	
FCA2	8D 0D DC	STA \$DC0D	Interruptmöglichkeiten löschen
FCA5	20 DD FD	JSR \$FDDD	CIA wieder auf Standardwerte, 1/60 s Timing
FCA8	AD A0 02	LDA \$02A0	
FCA8	F0 09	BEQ \$FCB6	
FCAD	8D 15 03	STA \$0315	IRQ auf Standard
FCB0	AD 9F 02	LDA \$029F	
FCB3	8D 14 03	STA \$0314	
FCB6	28	PLP	
FCB7	60	RTS	

*****			IRQ-Vektor setzen
FCB8	20 93 FC	JSR \$FC93	IRQ auf Standard
FCB8	F0 97	BEQ \$FC54	
FCBD	8D 93 FD	LDA \$FD93,X	
FCC0	8D 14 03	STA \$0314	IRQ-Vektor aus Tabelle setzen
FCC3	8D 94 FD	LDA \$FD94,X	
FCC6	8D 15 03	STA \$0315	
FCC9	60	RTS	

*****			
FCCA	A5 01	LDA \$01	
FCCE	09 20	ORA #\$20	Rekordermotor ausschalten
FCCE	85 01	STA \$01	
FCD0	60	RTS	

*****			prüft auf Erreichen der Endadresse
FCD1	38	SEC	
FCD2	A5 AC	LDA \$AC	laufende Adresse \$AC/\$AD
FCD4	E5 AE	SBC \$AE	
FCD6	A5 AD	LDA \$AD	Endadresse \$AE/\$AF
FCD8	E5 AF	SBC \$AF	
FCD4	60	RTS	

*****			Adresszeiger erhöhen
FCDB	E6 AC	INC \$AC	
FCDD	D0 02	BNE \$FCE1	
FCDF	E6 AD	INC \$AD	
FCE1	60	RTS	

*****			RESET
FCE2	A2 FF	LDX #\$FF	
FCE4	78	SEI	
FCE5	9A	TIS	
FCE6	D8	CLD	
FCE7	20 02 FD	JSR \$FD02	prüft auf ROM in \$8000
FCEA	D0 03	BNE \$FCEF	
FCEC	6C 00 B0	JMP (\$8000)	Sprung auf Modul-Start
FCEF	8E 16 D0	STX \$D016	Videocontroller Steuerregister 2
FCF2	20 A3 FD	JSR \$FDA3	Interrupt vorbereiten
FCF5	20 50 FD	JSR \$FD50	Arbeitsspeicher initialisieren
FCF8	20 15 FD	JSR \$FD15	Hardware und I/O Vektoren setzen
FCF8	20 5B FF	JSR \$FF5B	Video-Reset
FCFE	58	CLI	

```

FCFF  6C 00 A0  JMP ($A000)  zum BASIC Haltstart

***** prüft auf ROM in $8000
FD02  A2 05      LDX #$05
FD04  BD 0F FD    LDA $FD0F,X
FD07  DD 03 B0    CMP $B003,X  vergleicht mit 'CBMB0'
FD0A  D0 03      BNE $FD0F
FD0C  CA          DEY
FD0D  D0 F5      BNE $FD04
FD0F  60          RTS

***** ROM-Modul Identifizierung
FD10  C3 C2 CD 3B 30  'CBMB0'

***** Hardware und I/O Vektoren setzen/holen
FD15  A2 30      LDX #$30
FD17  A0 FD      LDY #$FD      Zeiger auf Tabelle $FD30
FD19  18          CLC
FD1A  B6 C3      STX $C3
FD1C  B4 C4      STY $C4
FD1E  A0 1F      LDY #$1F
FD20  B9 14 03    LDA $0314,Y
FD23  B0 02      BCS $FD27      C=1 dann Vektoren holen, C=0 setzen
FD25  B1 C3      LDA ($C3),Y
FD27  91 C3      STA ($C3),Y
FD29  99 14 03    STA $0314,Y
FD2C  BB          DEY
FD2D  10 F1      BPL $FD20
FD2F  60          RTS

***** Tabelle Hardware und I/O-Vektoren
FD30  31 EA 66 FE 47 FE 4A F3
FD38  91 F2 0E F2 50 F2 33 F3
FD40  57 F1 CA F1 ED F6 3E F1
FD48  2F F3 66 FE A5 54 ED F5

***** Arbeitsspeicher initialisieren
FD50  A9 00      LDA #$00
FD52  AB          TAY
FD53  99 02 00    STA $0002,Y  Zeropage,
FD56  99 00 02    STA $0200,Y  Page 2 und
FD59  99 00 03    STA $0300,Y  Page 3 löschen
FD5C  CB          INY
FD5D  D0 F4      BNE $FD53
FD5F  A2 3C      LDX #$3C
FD61  A0 03      LDY #$03
FD63  B6 B2      STX $B2      Bandspeicher Zeiger auf $033C
FD65  B4 B3      STY $B3
FD67  AB          TAY
FD68  A9 03      LDA #$03
FD6A  B5 C2      STA $C2      RAM ab $400 testen
FD6C  E6 C2      INC $C2
FD6E  B1 C1      LDA ($C1),Y
FD70  AA          TAX      Wert merken
FD71  A9 55      LDA #$55      101010101
FD73  91 C1      STA ($C1),Y
FD75  D1 C1      CMP ($C1),Y
FD77  D0 0F      BNE $FDBB
FD79  2A          ROL A      10101010
FD7A  91 C1      STA ($C1),Y

```



FD7C	D1 C1	CMP (\$C1),Y	
FD7E	DA 08	BNE \$FD88	
FD80	EA	TXA	Wert wieder zurückschreiben
FD81	51 C1	STA (\$C1),Y	
FD83	08	INY	
FD84	00 E8	BNE \$FD6E	
FD86	F0 E4	BEQ \$FD6C	
FD88	08	TYA	
FD89	AA	TAX	
FD8A	44 02	LDY \$C2	
FD8C	18	CLC	
FD8D	10 2D FE	JSR \$FE2D	Memory (RAM) Top setzen
FD90	47 08	LDA #\$08	
FD92	6D 82 02	STA \$0282	Memory (RAM) Start auf \$800
FD95	49 04	LDA #\$04	
FD97	5D 88 02	STA \$0288	Video-RAM auf \$400
FD9A	50	RTS	
***** IRQ Vektoren			
FD9B	54 FC CD FB 31 EA 2C F9	\$FC6A, \$FBCD, \$EA31, \$F92C	
***** Interrupt Initialisierung			
FDA3	49 7F	LDA #\$7F	Interrupt löschen
FDA5	6D 0D DC	STA \$DC0D	ICR CIA 1
FDA8	8D 0D DD	STA \$DD0D	ICR CIA 2
FDA9	6D 0D DC	STA \$DC0D	Port A CIA 1, Tastatur Matrixzeile 0
FDAE	49 08	LDA #\$08	
FDB0	6D 0E DC	STA \$DC0E	CRA CIA 1 Timer A 'one shot'
FDB3	6D 0E DD	STA \$DD0E	CRA CIA 2 Timer A 'one shot'
FDB6	8D 0F DC	STA \$DC0F	CRB CIA 1 Timer B 'one shot'
FDB9	6D 0F DD	STA \$DD0F	CRB CIA 2 Timer B 'one shot'
FDBC	42 00	LDX #\$00	Eingangs-Modus
FDBE	6E 03 DC	STX \$DC03	Datenrichtungsregister B CIA 1
FDC1	6E 03 DD	STX \$DD03	Datenrichtungsregister B CIA 2
FDC4	6E 18 D4	STX \$D418	Lautstärke für SID auf Null
FDC7	CA	DEX	Ausgabe-Modus
FDC8	9E 02 DC	STX \$DC02	Datenrichtungsregister A CIA 1
FDCB	A9 07	LDA #\$07	Videocontroller auf unterste 16 K
FDCD	8D 0D DD	STA \$DD0D	Port A CIA 2, ATN löschen
FDD0	A9 3F	LDA #\$3F	Bit 0 bis 5 auf Ausgabe
FDD2	8D 02 DD	STA \$DD02	Datenrichtungsregister A CIA 2
FDD5	A9 E7	LDA #\$E7	
FDD7	85 01	STA \$01	Prozessorport, Speicheraufteilung
FDD9	A9 2F	LDA #\$2F	
Fddb	85 00	STA \$00	Datenrichtung Prozessorport 0-5 Ausgang
FDDD	AD A6 02	LDA \$02A6	NTSC-Version ?
FDE0	F0 0A	BEQ \$FDEC	ja
FDE2	A9 25	LDA #\$25	
FDE4	8D 04 DC	STA \$DC04	Timer für PAL-Version setzen
FDE7	A9 40	LDA #\$40	\$4025 = 16421 Zyklen
FDE9	4C F3 FD	JMP \$FDF3	
FDEC	A9 95	LDA #\$95	
FDEE	8D 04 DC	STA \$DC04	Timer für NTSC-Version setzen
FDF1	A9 42	LDA #\$42	\$4295 = 17045 Zyklen
FDF3	8D 05 DC	STA \$DC05	Timer high
FDF6	4C 6E FF	JMP \$FF6E	Interrupt durch Timer setzen
***** Parameter für Filenamen setzen			
FDF9	85 B7	STA \$B7	Länge
FDFB	86 B8	STX \$B8	Adresse low

FDFD	84 BC	STY \$BC	Adresse high
FDFE	60	RTS	

*****			Parameter für aktives File setzen
FE00	85 B8	STA \$B8	logische Filenummer
FE02	86 BA	STX \$BA	Geräte Adresse
FE04	84 B9	STY \$B9	Sekundär Adresse
FE06	60	RTS	

*****			Status holen
FE07	A5 BA	LDA \$BA	Gerätenummer
FE09	C9 02	CMP #\$02	gleich 2 ?
FE0B	D0 0D	BNE.\$FE1A	nein
FE0D	AD 97 02	LDA \$0297	RS232 Status holen
FE10	4B	PHA	
FE11	A9 00	LDA #\$00	Status löschen
FE13	BD 97 02	STA \$0297	
FE16	6B	PLA	
FE17	60	RTS	

*****			Flag für Betriebssystemmeldungen setzen
FE1B	85 90	STA \$90	
FE1A	A5 90	LDA \$90	

*****			Status setzen
FE1C	05 90	DRA \$90	
FE1E	85 90	STA \$90	
FE20	60	RTS	

*****			Timeout-Flag für IEC setzen
FE21	BD B5 02	STA \$02B5	
FE24	60	RTS	

*****			MEMTOP Obergrenze BASIC-Ram holen/setzen
FE25	90 06	BCC \$FE2D	
FE27	AE B3 02	LDX \$02B3	Carry gesetzt
FE2A	AC B4 02	LDY \$02B4	Adresse nach X/Y holen
FE2D	BE B3 02	STX \$02B3	Carry gelöscht
FE30	BC B4 02	STY \$02B4	X/Y nach Adresse setzen
FE33	60	RTS	

*****			MEMBOT Untergrenze BASIC-RAM holen/setzen
FE34	90 06	BCC \$FE3C	
FE36	AE B1 02	LDX \$02B1	s.o.
FE39	AC B2 02	LDY \$02B2	
FE3C	BE B1 02	STX \$02B1	
FE3F	BC B2 02	STY \$02B2	
FE42	60	RTS	

*****			NMI Einsprung
FE43	7B	SEI	
FE44	6C 1B 03	JMP (\$031B)	JMP \$FE47, NMI-Vector
FE47	4B	PHA	
FE48	8A	TXA	
FE49	4B	PHA	Register retten
FE4A	9B	TYA	
FE4B	4B	PHA	
FE4C	A9 7F	LDA #\$7F	
FE4E	BD 0D DD	STA \$DDDD	Interruptmöglichkeiten löschen
FE51	AC 0D DD	LDY \$DDDD	Flags lesen und löschen

FE54	30 1C	BMI \$FE72	RS 232 aktiv ?
FE56	20 02 FD	JSR \$FD02	Prüft auf ROM-Modul in \$8000
FE59	D0 03	BNE \$FE5E	nein, weiter
FE5B	6C 02 80	JMP (\$8002)	ja, Sprung auf Modul-NMI
FE5E	20 BC F6	JSR \$F6BC	Flag für Stop-Taste setzen
FE61	20 E1 FF	JSR \$FFE1	Stop-Taste abfragen
FE64	D0 0C	BNE \$FE72	nicht gedrückt
FE66	20 15 FD	JSR \$FD15	Standard-Vektoren für Interrupt und I/O setzen
FE69	20 A3 FD	JSR \$FDA3	I/O initialisieren
FE6C	20 18 E5	JSR \$E518	I/O initialisieren und Bildschirm löschen
FE6F	6C 02 A0	JMP (\$A002)	zum BASIC-Warmstart

\*\*\*\*\* NMI-Routine für RS 232

FE72	98	TXA	
FE73	2D A1 02	AND \$02A1	
FE76	AA	TAX	
FE77	29 01	AND #\$01	
FE79	F0 28	BEQ \$FEA3	
FE7B	AD 00 DD	LDA \$DD00	
FE7E	29 FB	AND #\$FB	
FE80	05 B5	ORA \$B5	
FE82	8D 00 DD	STA \$DD00	
FE85	AD A1 02	LDA \$02A1	
FE88	8D 00 DD	STA \$DD00	
FE8B	8A	TXA	
FE8C	29 12	AND #\$12	
FE8E	F0 0D	BEQ \$FE9D	
FE90	29 02	AND #\$02	
FE92	F0 06	BEQ \$FE9A	
FE94	20 D6 FE	JSR \$FED6	RS 232 in
FE97	4C 9D FE	JMP \$FE9D	
FE9A	20 07 FF	JSR \$FF07	RS 232 out
FE9D	20 BB EE	JSR \$EEBB	RS 232 Ausgabe
FEA0	4C B6 FE	JMP \$FEB6	Rückkehr vom Interrupt
FEA3	8A	TXA	
FEA4	29 02	AND #\$02	
FEA6	F0 06	BEQ \$FEAE	
FEA8	20 D6 FE	JSR \$FED6	RS 232 in
FEAB	4C B6 FE	JMP \$FEB6	Rückkehr vom Interrupt
FEAE	8A	TXA	
FEAF	29 10	AND #\$10	
FEB1	F0 03	BEQ \$FEB6	
FEB3	20 07 FF	JSR \$FF07	RS 232 out
FEB6	AD A1 02	LDA \$02A1	
FEB9	8D 00 DD	STA \$DD00	
FEBC	68	PLA	
FEBD	A8	TAY	
FEBE	68	PLA	Register zurückholen
FEBF	AA	TAX	
FEC0	68	PLA	
FEC1	40	RTI	

*****		Timerkonstanten für RS 232 Baud-Rate, NTSC-Version	
FEC2	C1 27	\$27C1	= 10177      50 Baud
FEC4	3A 1A	\$1A3E	= 6718        75 Baud
FEC6	C5 11	\$11C5	= 4549        110 Baud
FEC8	74 0E	\$0E74	= 3700        134.5 Baud
FECA	ED 0C	\$0CED	= 3309        150 Baud
FEC C	45 06	\$0645	= 1605        300 Baud

FECF	F0 02	\$02F0 =	752	600 Baud
FED0	46 01	\$0146 =	326	1200 Baud
FED2	B9 00	\$00B9 =	184	1800 Baud
FED4	71 00	\$0071 =	113	2400 Baud

```
***** NMI-Routine für RS 232 Eingabe
FED6 AD 01 DD LDA $DD01      Port Register B
FED9 29 01    AND #$01      Bit für Receive Data isolieren
FEDB B5 A7    STA $A7
FEDD AD 06 DD LDA $DD06
FEE0 E9 1C    SBC #$1C
FEE2 6D 99 02 ADC $0299
FEE5 BD 06 DD STA $DD06
FEEB AD 07 DD LDA $DD07      RS 232 Timerkonstanten für Baud-Rate
FEEB 6D 9A 02 ADC $029A
FEEE BD 07 DD STA $DD07      in Timer schreiben
FEF1 A9 11    LDA #$11
FEF3 BD 0F DD STA $DD0F      Control Register B
FEF6 AD A1 02 LDA $02A1
FEF9 BD 0D DD STA $DD0D      Interrupt Control Register
FEFC A9 FF    LDA #$FF
FEFE BD 06 DD STA $DD06
FF01 BD 07 DD STA $DD07      Timer setzen
FF04 4C 59 EF JMP $EF59      Bit holen
```

```
***** NMI-Routine für RS 232 Ausgabe
FF07 AD 95 02 LDA $0295
FF0A BD 06 DD STA $DD06
FF0D AD 96 02 LDA $0296      RS 232 Timerkonstanten für Baud-Rate
FF10 BD 07 DD STA $DD07
FF13 A9 11    LDA #$11
FF15 BD 0F DD STA $DD0F      Control Register B
FF18 A9 12    LDA #$12
FF1A 4D A1 02 EOR $02A1
FF1D BD A1 02 STA $02A1
FF20 A9 FF    LDA #$FF
FF22 BD 06 DD STA $DD06
FF25 BD 07 DD STA $DD07      Timer laden
FF28 AE 98 02 LDX $0298      Anzahl der zu sendenden Bits
FF2B B6 AB    STX $AB
FF2D 60      RTS
```

```
*****
FF2E AA      TAX
FF2F AD 96 02 LDA $0296
FF32 2A      ROL A
FF33 AB      TAY
FF34 BA      TXA
FF35 69 CB    ADC #$CB
FF37 BD 99 02 STA $0299      RS 232 Timing
FF3A 98      TYA
FF3B 69 00    ADC #$00
FF3D BD 9A 02 STA $029A
FF40 60      RTS

FF41 EA      NOP
FF42 EA      NOP
```

```
***** Einsprung aus Bandroutine
```

FF43	08	PHP	
FF44	6B	PLA	
FF45	29 EF	AND #EF	Break-Flag löschen
FF47	4B	PHA	
*****			IRQ-Einsprung
FF48	4B	PHA	
FF49	8A	TXA	
FF4A	4B	PHA	Register retten
FF4B	9B	TYA	
FF4C	4B	PHA	
FF4D	BA	TSX	
FF4E	BD 04 01	LDA \$0104,x	Break-Flag vom Stapel holen
FF51	29 10	AND #10	und testen
FF53	F0 03	BEQ \$FF5B	nicht gesetzt
FF55	6C 16 03	JMP (\$0316)	BREAK - Routine
FF5B	6C 14 03	JMP (\$0314)	Interrupt - Routine
*****			Video-Reset
FF5B	20 18 E5	JSR \$E518	Videocontroller initialisieren
FF5E	AD 12 D0	LDA \$D012	Rasterzeile
FF61	D0 FB	BNE \$FF5E	wartet auf Ende Videoreihe
FF63	AD 19 D0	LDA \$D019	Interrupt durch Rasterzeile ?
FF66	29 01	AND #01	
FF6B	BD A6 02	STA \$02A6	PAL/NTSC-Version merken
FF6B	4C DD FD	JMP \$FDDDD	Interrupttimer setzen
*****			Timer für Interrupt setzen
FF6E	A9 B1	LDA #B1	Timer A Unterlauf
FF70	BD 0D DC	STA \$DC0D	Interrupt Control Register
FF73	AD 0E DC	LDA \$DC0E	Control Register A
FF76	29 B0	AND #B0	Bit 7 löschen, Uhr mit 60 Hz triggern
FF7B	09 11	ORA #11	Timer A starten
FF7A	BD 0E DC	STA \$DC0E	Timer A starten, Force Load
FF7D	4C 8E EE	JMP \$EE8E	seriellen Takt aus
FFB0	00	BRK	
*****			Sprungtabelle für Betriebssystem-Routinen
FFB1	4C 5B FF	JMP \$FF5B	Video-Reset
FFB4	4C A3 FD	JMP \$FDA3	CIA's initialisieren
FFB7	4C 50 FD	JMP \$FD50	RAM löschen bzw. testen
FFBA	4C 15 FD	JMP \$FD15	I/O initialisieren
FFBD	4C 1A FD	JMP \$FD1A	I/O Vektoren initialisieren
FF90	4C 18 FE	JMP \$FE18	Status setzen
FF93	4C B9 ED	JMP \$EDB9	Sekundär-Adresse nach LISTEN senden
FF96	4C C7 ED	JMP \$EDC7	Sekundär-Adresse nach TALK senden
FF99	4C 25 FE	JMP \$FE25	RAM-Ende setzen/holen
FF9C	4C 34 FE	JMP \$FE34	RAM-Anfang setzen/holen
FF9F	4C B7 EA	JMP \$EAB7	Tastatur abfragen

FFA2	4C 21 FE	JMP \$FE21	Time-out-Flag für IEC-Bus setzen
FFA5	4C 13 EE	JMP \$EE13	Eingabe vom IEC-Bus
FFA8	4C DD ED	JMP \$EDDD	Ausgabe vom IEC-Bus
FFAB	4C EF ED	JMP \$EDEF	UNTALK senden
FFAE	4C FE ED	JMP \$EDFE	UNLISTEN senden
FFB1	4C 0C ED	JMP \$ED0C	LISTEN senden
FFB4	4C 09 ED	JMP \$ED09	TALK senden
FFB7	4C 07 FE	JMP \$FE07	Status holen
FFBA	4C 00 FE	JMP \$FE00	Fileparameter setzen
FFBD	4C F9 FD	JMP \$FDF9	Filenameparameter setzen
FFC0	6C 1A 03	JMP (\$031A)	\$F34A OPEN
FFC3	6C 1C 03	JMP (\$031C)	\$F291 CLOSE
FFC6	6C 1E 03	JMP (\$031E)	\$F20E CHKIN Eingabegerat setzen
FFC9	6C 20 03	JMP (\$0320)	\$F250 CKOUT Ausgabegerat setzen
FFCC	6C 22 03	JMP (\$0322)	\$F333 CLRCH Ein/Ausgabe rucksetzen
FFCF	6C 24 03	JMP (\$0324)	\$F157 BASIN Eingabe eines Zeichens
FFD2	6C 26 03	JMP (\$0326)	\$F1CA BSOUT Ausgabe eines Zeichens
FFD5	4C 9E F4	JMP \$F49E	LOAD
FFD8	4C DD F5	JMP \$F5DD	SAVE
FFDB	4C E4 F6	JMP \$F6E4	Time setzen
FFDE	4C DD F6	JMP \$F6DD	Time holen
FFE1	6C 2B 03	JMP (\$032B)	\$F6ED STOP-Taste abfragen
FFE4	6C 2A 03	JMP (\$032A)	\$F13E GET
FFE7	6C 2C 03	JMP (\$032C)	\$F32F CLALL
FFEA	4C 9B F6	JMP \$F69B	Time erhöhen
FFED	4C 05 E5	JMP \$E505	SCREEN Anzahl Zeilen/Spalten holen
FFF0	4C 0A E5	JMP \$E50A	Cursor setzen / Cursorposition holen
FFF3	4C 00 E5	JMP \$E500	Startadresse des I/O-Bausteins holen
FFF6	52 52 42 59		

\*\*\*\*\* Hardware Vektoren

FFFA	43 FE	\$FE43	NMI Vektor
FFFC	E2 FC	\$FCE2	RESET Vektor
FFFE	4B FF	\$FF4B	IRQ Vektor

Zu Anfang dieses Kapitels einige Vorbemerkungen:  
Leider können die folgenden Seiten keine Einführung in die Digital- oder Computertechnik bieten.  
Wir müssen einige elementare Kenntnisse dieser Technik voraussetzen. So sollten Sie den Unterschied zwischen einem AND- und einem OR-Gate kennen, oder sich beispielsweise in der Benutzung der Hexadezimalzahlen auskennen.  
Andernfalls ist die Lektüre in die Digitaltechnik einführender Literatur angeraten.

Wenn Sie diese Grundkenntnisse bereits haben, bisher aber mit der Hardware von Microcomputern nichts zu tun hatten, so sollten Sie sich von der etwas verwirrenden Anzahl der Leitungen, Gatter und anderen ICs im Schaltplan nicht beeindrucken lassen.  
Nach der Lektüre dieses Kapitels werden Sie die Hardware Ihres Computers recht gut verstehen.

Den Spezialisten und 'Freaks' unter Ihnen wird die Beschreibung sicher zu ausführlich erscheinen.  
Sie sollten dies Kapitel trotzdem in Ruhe durchlesen. Um die Funktionen der einzelnen Stufen nur an Hand des Schaltplanes im Detail zu verstehen, ist wesentlich mehr Zeit erforderlich, als zum Lesen dieses Kapitels benötigt wird.

Schauen Sie doch mal rein.

Jeder technisch interessierte Computerbesitzer hat sicher den Wunsch, sein Gerät einmal zu öffnen und hineinzuschauen.  
Vielleicht haben auch Sie schon einmal das Innenleben betrachtet. Sollten Sie aber aus Vorsicht, den CBM 64 nicht zu beschädigen diesen Wunsch nicht nachgegeben haben, dann seien Sie beruhigt.  
Lösen Sie zuerst alle Leitungen zum CBM 64, also Netzteil, Fernseher und alle anderen angeschlossenen Geräte. Dann können Sie unbesorgt zu einem passenden Kreuzschlitzschraubenzieher greifen, die auf der Unterseite befindlichen drei Schrauben lösen und vorsichtig die beiden Gehäusenhälften trennen um einen Blick in den Computer zu werfen.  
Sie sollten aber um ganz sicher zu gehen vorher alle Steckverbindungen lösen.  
Dann kann nichts passieren und Sie können sich einen Eindruck von dem machen, was Gegenstand der kommenden Seiten sein soll.



## Die Stromversorgung.

Obwohl die Stromversorgung zu den einfachen Schaltkreisen in einem Computer gehört, hat der Entwickler doch einige Tricks angewendet, um mit minimalem Aufwand einen größtmöglichen Effekt zu erzielen.

Den Anschluß an das Lichtnetz übernimmt der Trafo. Dieser Trafo befindet sich zusammen mit einer Gleichrichterschaltung im Trafogehäuse und wird über einen 7-poligen DIN-Stecker an die mit CN7 bezeichnete Buchse angeschlossen. Im Trafo wird eine Wechselspannung von 9 Volt erzeugt, die auf die Pins 6 und 7 von CN7 geführt werden. Die Gleichrichterschaltung im Trafogehäuse erzeugt über eine zweite Trafowicklung eine stabilisierte Gleichspannung von 5 Volt. Diese 5 Volt liegen auf dem Pin 5 von CN7, die Masseleitung auf den Pins 1, 2 und 3.

Die von den Buchsenkontakten kommenden Spannungen werden zur Beseitigung von Netzstörungen über die Spulen L2 und L4 und die Kondensatoren C20, C21, C98, C99 und C100 geführt und gefiltert.

Der mit SW1 bezeichnete doppelpolige Schalter ist der an der rechten Seite befindliche Einschalter.

Die 9 Volt Wechselspannung wird mit der Sicherung F1 (1 Ampere) abgesichert und steht am Userport an den Kontakten 11 und 12 zur Verfügung. Diese Spannung können Sie nach Gleichrichtung und Siebung für externe Geräte verwenden, belasten Sie diese Stromquelle jedoch nur mit maximal 100 mA, die Sicherung wird es Ihnen danken.

### Apropos Sicherung:

Wenn sie defekt ist, leuchtet die LED an 64, auch eine angeschlossene Floppy macht einen Reset, auf dem Bildschirm ist aber nichts zu sehen.

Vergewissern Sie sich aber zuerst, ob der Fernseher auf den richtigen Kanal steht und das HF-Kabel angeschlossen ist.

Wenn alles richtig erscheint, kontrollieren Sie die Sicherung. Ist diese durchgebrannt, dann ersetzen Sie sie durch eine Sicherung mit dem Wert 1.25 Ampere. Dieser Wert ist noch absolut unkritisch und hat den Vorteil, daß die Sicherung hält.

Sollte auch diese ihren 'Geist' aufgeben, liegt mit einiger Wahrscheinlichkeit ein Defekt vor.

Nach der Sicherung kommt eine Gleichrichterschaltung, welche 5 Volt stabilisiert, 9 Volt unreguliert sowie 12 Volt stabilisiert zur Verfügung stellt.

Die Gleichrichterschaltung besteht aus dem Brückengleichrichter CR4 und den Dioden CN5 und CN6. Hinter dem Brückengleichrichter stehen die unregulierten 9 Volt, die mit VR2, einem integrierten 5V-Festspannungsregler, auf 5 Volt stabilisiert werden.

Über die Dioden CN5 und CN6 wird die Wechselspannung auf eine unregulierte Gleichspannung von ca. 16 Volt gleichgerichtet, die mit dem Spannungsregler VR1 auf 12 Volt stabilisiert wird.

Die aus dem Trafogehäuse kommenden 5 Volt sind mit einem eigenen Spannungsregler schon im Trafogehäuse stabilisiert. Dies hat den Vorteil, das die erzeugte Verlustwärme nicht den Computer aufheizt, der erzeugt schon genug eigene Wärme. Diese Spannung übernimmt die Versorgung der meisten ICs in Ihrem CBM 64 und liegt am Pin 2 des Userports CN2. Damit steht Ihnen für kleinere Projekte schon eine geeignete Spannung zur Verfügung. Aber auch diese Spannungsquelle sollten Sie nicht überlasten. Der Maximalstrom ist mit 100 mA angegeben, sicherlich für einige ICs ausreichend. Erfreulicherweise ist diese Spannung kurzzeitig kurzschlußfest. Dieser Kurzschlußfall ist sehr einfach feststellbar, in diesem Fall erscheint kein Bild auf dem angeschlossenen Fernseher und die Leuchtdiode leuchtet nicht, da auch sie von dieser Spannung versorgt wird.

Die im CBM 64 erzeugte Spannung von 5 Volt trägt die Bezeichnung CAN+5. Diese Spannung versorgt den Videocontroller (weiterhin kurz als VIC bezeichnet), die Videoausgangsstufe und alle zur Takterzeugung benötigten ICs. Der VIC bekommt die 5 Volt direkt, zur Videoausgangsstufe wird die Spannung über die Spule L1 und die Kondensatoren C61, C63 und C64 gefiltert. Alle der Takterzeugung zugehörigen Bauteile bekommen die Spannung über L2, C65, C66 und C67 gesiebt zugeführt.

Da die Datasette kein eigenes Netzteil hat, muß der Computer auch den benötigten 'Baft' hierfür liefern. Die von der Datasette benötigten Spannungen sind 6 Volt für den Recordermotor und 5 Volt für die eingebaute Elektronik. Der Antriebsmotor bekommt die Spannung über die Transistorschaltung Q1, Q2 und Q3 auf die Kontakte 3 und C des Kassettenportsteckers CN3 geschaltet. Wenn der Prozessor das Portbit 5 auf High legt, wird der Transistor Q2 durchgeschaltet. Damit ist die Zenerdiode CR2 kurzgeschlossen, der Transistor Q1 bekommt keine Basisvorspannung, Q1 und Q3 sperren. Der Recordermotor stoppt. Wird das Portbit dagegen Low, dann ist der Transistor Q2 gesperrt. An der Basis von Q1 liegt die Zenerspannung von 7.5 Volt und steuert die Transistoren Q1 und Q3 an. Am Emitter von Q3 liegt die um die beiden Basis-Emitterspannungen der Transistoren (ca 1.5 Volt) reduzierte Zenerspannung, das ergibt ca. 6 Volt. Durch diese Stabilisierung der Motorschaltstufe wird eine konstante Drehzahl des Motors erreicht. Die Elektronik der Datasette wird über die Kontakte 2 und B des Steckers CN3 versorgt.

Bleiben noch die 12 Volt. Diese Spannung wird für den VIC, den SID (Sound Interface Device) und die Audioausgangsstufe mit dem Transistor Q8 benötigt.

Nicht direkt zur Stromversorgung gehörend ist die kleine

Schaltung rund um das Gatter U27. Trotzdem soll sie hier erläutert werden, da sie ihre Signale aus dem Netzteil bekommt.

Das Gatter U27 stellt eine UND-Verknüpfung dar. Der Eingang Pin 13 liegt fest an 5 Volt, der Eingang Pin 12 über den Widerstand R5 an den 9 Volt Wechselspannung. Am Pin 12 würde sich die Spannung also mit der Netzfrequenz von 50 Hertz ändern.

Nun ist eine Spannung von 9 Volt für einen TTL-Eingang nicht sehr verträglich und eine negative Spannung von -9 Volt sollte an einem solchen Eingang unbedingt vermieden werden, um das IC nicht zu zerstören.

Um die Eingangsspannung zu begrenzen, ist die Zenerdiode CR1 an den Eingang geschaltet.

Wenn die Wechselspannung über +2,7 Volt steigt, so wird sie von der Zenerdiode auf diesen Wert begrenzt. Damit ist ein logisches High-Signal gegeben.

Die negative Spannung wird von der Zenerdiode auf -0,7 Volt begrenzt, ein Wert, den der TTL-Eingang noch gut verkraftet, und der als logisches Low-Signal erkannt wird.

Die Spannung schwankt also im Rhythmus der Netzfrequenz am Pin 12 des U27 zwischen Low und High. Damit ändert sich der Ausgang im selben Takt.

Der Widerstand R37 stellt eine Mitkopplung dar, er beschleunigt die Anstiegs- und Abfallzeiten, um saubere Rechteckimpulse für die weitere Verwendung zur Verfügung zu stellen.

Woraus besteht nun die weitere Verwendung?

Im Schaltplan kann man erkennen, dass diese 50 Hertz an die ICs U1 und U2, die beiden CIAs, gehen.

Auf die CIAs wird im weiteren Verlauf der Schaltplanbeschreibung noch näher eingegangen. Jetzt nur so viel:

Die Netzfrequenz ist das am einfachsten zu erzeugende frequenzkonstante Signal. Darum eignet es sich besonders für Anwendungen, in denen Zeiten gemessen werden sollen. Das ist auch Aufgabe des Signals in den CIAs. Diese enthalten sogenannte Echtzeituhren, die ihren Takt von der Netzfrequenz beziehen.

## Die Takterzeugung.

Für ein ordnungsgemäßes Funktionieren eines Computers ist eine stabile und störungsfreie Stromversorgung sehr wichtig. Die Konstanz und Stabilität der Taktsignale ist für die Funktion aber sicher genau so maßgebend. Dieser Takterzeugung wollen wir uns jetzt zuwenden.

Wenn Sie auf die Leiterplatte des CBM 64 schauen und mit dem Schaltplan auf IC-Suche gehen, so werden Sie vermutlich das eine oder andere IC nicht auf den ersten Blick finden. So sicher auch die für die Taktversorgung zuständigen ICs. Diese befinden sich zusammen mit dem VIC (Video Interface Controller) in dem Blechkasten in der Mitte der Platine (nicht der Kasten mit dem Fernsehanschluß, das ist der UHF-Modulator).

Dieses Blechgehäuse schirmt die bei der Takterzeugung entstehende hochfrequente Störstrahlung ab. Bei Rechnern ohne ausreichende Abschirmung kann man beobachten, daß alle im näheren Umkreis befindlichen Radios nur Pfeif- und Zischlaute von sich geben. Schlimmer noch, auch Fernsehgeräte werden von solchen Störstrahlungen beeinflusst. Wenn der 64 nicht über ausreichende Entstörmaßnahmen verfügen würde, wäre der Betrieb mit einem Fernseher wenn auch nicht unmöglich, so doch sehr gestört.

Die alles bestimmende Taktfrequenz wird vom Quarz Y1 erzeugt. Doch vorab noch eine Erläuterung. Alle jetzt folgenden Angaben beziehen sich auf ein für den deutschen Markt produziertes Gerät mit PAL-Ausgang.

Der Quarz Y1 schwingt mit einer Frequenz von 17.734472 MHz. Er ist über C70 an das IC U31 angeschlossen. Das IC U31, ein TTL-IC mit der Bezeichnung 74LS629, enthält 2 unabhängige VCOs.

Ein VCO ist ein spannungsgesteuerter Oszillator. Durch eine an Steuereingang angelegte Gleichspannung kann die Frequenz in einem bestimmten Bereich verändert werden. Dieser Steuereingang ist für den VCO 1 der Pin 1. Das Poti R27 an diesem Eingang erlaubt eine wenn auch geringfügige Änderung der Ausgangsfrequenz. Da auch Quarze eine gewisse Toleranz haben, läßt sich die Soll-Frequenz mit dem Poti genau einstellen.

Der Ausgang des VCO 1 ist der Pin 10. Die hier anliegende Frequenz wird direkt als Signal 0COLOR an den VIC geführt. Gleichzeitig gelangt das Signal an das IC U30. Dieses IC, ein 74LS193, ist als Frequenzteiler geschaltet. Dieser Teiler hat ein einstellbares Teilerverhältnis. In Abhängigkeit der Pegel an den Pins 1, 9, 10 und 15 läßt sich jedes Teilerverhältnis zwischen 1:1 und 15:1 einstellen.

In unserem Fall ist das Teilerverhältnis auf 9:1 eingestellt. Die 17.734 MHz werden also durch 9 geteilt. Damit steht am

Ausgang Pin 6 eine Frequenz von 1.9784 MHz zur Verfügung. Diese Frequenz wird auf den Pin 11 des IC U29 geführt. U29 enthält 2 Flipflops. Mit jeder positiven Flanke des Clock-Signals an Pin 11 wird die am Dateneingang Pin 12 des Flipflops 1 liegende Information auf den Q-Ausgang Pin 9 weitergegeben. Der Ausgang -Q (Pin 8) hat dann auch die Eingangsinformation, nur mit invertierter Polarität. In der vorliegenden Beschaltung liefert die durch 9 geteilte Quarzfrequenz das Clocksignal für FF1. Der Dateneingang ist mit dem Ausgang -Q verbunden. Wenn dieser -Q-Ausgang High ist, wird das High-Signal mit der nächsten positiven Flanke an Pin 11 auf den Q-Ausgang gegeben. Gleichzeitig wird der -Q-Ausgang Low. Mit der nächsten positiven Taktflanke wird das Low an Q gelegt, -Q hat jetzt wieder ein High und so weiter. Besser sieht man diese Vorgänge im Bild auf der nächsten Seite. Hier sind die Frequenzen und Phasenlagen dargestellt.

Mit jedem zweiten Taktimpuls wechseln also die Ausgänge ihren Zustand. Das kommt einer Frequenzteilung durch den Faktor 2 gleich, am Ausgang erscheint eine Frequenz von 985,248 KHz. Das ist die Taktfrequenz des Prozessors. Dieses Signal wird aber nicht direkt als Takt verwendet, die ganze Sache ist etwas komplizierter. Das Signal Dot Clock mit der Frequenz 7,88198 MHz läßt sich durch Frequenzteilung nicht aus der Quarzfrequenz ableiten. Darum muß ein anderer Weg beschritten werden, die Frequenzsynthese mit einer PLL-Schaltung. PLL bedeutet Phase Locked Loop, übersetzt etwa Phasengeregelte Schleife. Der PLL im 64 ist mit den ICs U32, U31 und dem VIC aufgebaut.

Wichtigster Bestandteil eines PLL ist ein Phasencomparator mit zwei Eingängen. Dieser Phasencomparator liefert an seinem Ausgang eine Gleichspannung, die proportional der Phasenlage der beiden Signale ist. Diese Funktion ist mit dem IC U32 und dem Transistor Q7 aufgebaut.

Im Detail funktioniert die Sache etwa so:

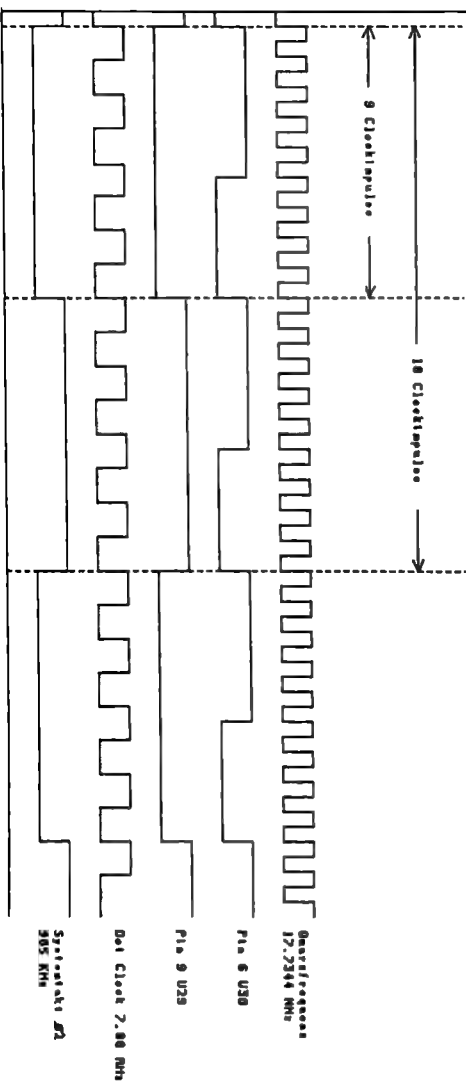
Am Eingang Pin 1 des U32 liegt eine Frequenz von 985 KHz, geliefert vom Ausgang des Flipflop U29. Am zweiten Eingang des PLL Pin 3 liegt das Signal 80, das vom VIC gelieferte Taktsignal für den Prozessor, mit noch unbestimmter Frequenz.

Dieses Signal 80 vom VIC stellt das durch 8 geteilte Ausgangssignal des VCO 2 im U31 dar. Diese Frequenzteilung durch 8 findet direkt im VIC statt.

Die Frequenz des VCO 2 wird nicht durch einen Quarz sondern durch einen Kondensator, den C86, bestimmt. Die Steuerspannung des VCO 2 wird jetzt durch den Ausgang des Phasencomparators U32 geliefert.

Wenn die Steuerspannung des VCO 2 ca. 3 Volt beträgt, schwingt er auf einer Frequenz von 7,88198 MHz.

# Die Taktsignale und ihre Phasenlage



Wenn wir jetzt den Fall annehmen, daß die Frequenz des Flipflops U29 höher als die Frequenz 00 ist, der VCO 2 also beispielsweise nur mit 7.7 MHz schwingt, dann liefert der Ausgang Pin 8 des Phasencomparators eine Spannung kleiner 3 Volt, die den VCO mit einer höheren Frequenz schwingen läßt. Damit erhöht sich auch die Frequenz am Pin 3 der Phasencomparators, sie nähert sich der Referenzfrequenz am Pin 1, die Steuerspannung nähert sich den 3 Volt. Wenn die Frequenzen an Pin 1 und Pin 3 gleich sind, wird der VCO noch so lange geregelt, bis die Signale nicht nur frequenz- sondern auch phasengleich sind. Der selbe Vorgang läuft ab, wenn der VCO mit zu hoher Frequenz schwingt. Dann wird die Steuerspannung größer 3 Volt. Jetzt schwingt der VCO langsamer und die Steuerspannung nimmt ab, bis die Signale frequenz- und phasengekoppelt sind. Dann liegt das Signal Dot Clock richtig an. Die geschilderten Regelvorgänge brauchen nur kurze Zeit. Nach spätestens 100 Millisekunden stehen alle Frequenzen zur Verfügung.

Zum Abschluss noch eine kurze Schilderung der Funktion des FF2 und der Abläufe in einem 64 mit NTSC-Farbausgang. In diesen für den amerikanischen Markt produzierten Geräten ist zum einen 14.31818 MHz-Quarz eingebaut. Des weiteren ist ein anderer VIC-Chip, ein 6567, in der NTSC-Version eingesetzt. Bei der PAL-Version ist dies ein 6569.

Als drittes Merkmal ist die Drahtbrücke zwischen den Punkten E1 und E2 oder E3 anders gelegt. Bei Pal-Geräten ist diese Brücke zwischen E1 und E2 geschaltet. Damit liegen die Pins 1 und 10 des Teilers U30 an +5 Volt. Auch der Pin 4 des IC U29 liegt an High.

Dieser Pin 4 ist der sogenannte Preset-Eingang an FF2. Clock-, Daten- und Clear-Eingang dieses FFs sind an Masse gelegt. Ein Low-Signal am Clear-Eingang versetzt das Flipflop in einen definierten Zustand. Unabhängig von den anderen Eingangssignalen wird der Q-Ausgang Low, -Q dagegen High. Wie bei so vielen anderen Gelegenheiten gibt es aber eine wichtige Einschränkung zu dem zuvor Gesagten. Der Preset-Eingang muss dafür High-Pegel haben. Diese Bedingung ist bei einem PAL-Gerät über die Drahtbrücke erfüllt.

Die Eingänge 1, 9, 10 und 15 des Zählers U30 bestimmen binär codiert den Startwert des Zählers. Da der Zähler immer bis 16 zählt, kann man mit dem Startwert das Teilverhältnis einstellen. Er beginnt dann nicht bei 0, sondern mit dem programmierten Wert.

Der Eingang A stellt das niederwertige Bit dar, Eingang D das höchstwertige Bit. An diesen Eingängen liegt dezimal ausgedrückt eine 7. Der Zähler zählt bis 16 weiter, und beginnt dann wieder bei 7. Für diesen Durchlauf benötigt er 9 Zählimpulse, er teilt also durch 9.

Damit stellt das FF2 nichts anderes als einen einfachen Inverter dar. Wenn der Eingang High ist, so ist der Ausgang Low und umgekehrt, die normale Inverterfunktion.

Bei NTSC liegt der Preset-Eingang des U29 auf Low. Laut

Datenblatt haben jetzt sowohl Q- wie auch -Q-Ausgang High-Pegel, eigentlich ein ungewöhnlicher Zustand, der das IC aber nicht beschädigt.

Jetzt ist das Teilerverhältnis von U30 7:1, mit dem nachfolgenden Flipflop 14:1 und die Taktfrequenz des Prozessors beträgt damit 1.0227 MHz, geringfügig höher als die PAL-Arbeitsfrequenz.



## Der Prozessor

Wie schon erwähnt ist der Prozessor des CBM 64 der 6510. Dieser neue Prozessor unterscheidet sich von dem bekannten 6502 in der Hauptsache durch einen im Prozessorchip integrierten Port. Dieser Port verfügt über 6 programmierbare IO-Leitungen (IO = Abkürzung für Input Output; Leitungen, die wahlweise als Eingänge oder Ausgänge geschaltet sind). Die Zahl 6 ist im Zusammenhang mit 8-Bit-Prozessoren sicher etwas ungewöhnlich. Bei dem zur Verfügung stehenden 40-poligen Gehäuse waren aber nicht mehr Leitungen frei, um einen vollen 8-Bit-Port zu realisieren.

Die 40 Pins des 6510 sind wie folgt belegt:

Pin	Bez.	Funktion
1	0IN	Eingang, Systemtakt vom VIC Pin 17
2	RDY	Eingang, Ready von U27 Pin 3
3	-IRQ	Eingang, Interrupt Request
4	-NMI	Eingang, Non Maskable Interrupt
5	AEC	Eingang, Address Enable Control
6	VCC	Betriebsspannung +5V
7	A0	Ausgang, Adressbit 0
bis		
20	A13	Ausgang, Adressbit 13
21	GND	Betriebsspannung Masse
22	A14	Ausgang, Adressbit 14
23	A15	Ausgang, Adressbit 15
24	PB5	Ein-Ausgang, Portbit 5
bis		
29	PB0	Ein-Ausgang, Portbit 0
30	D7	Ein-Ausgang, Datenbit 7
bis		
37	D0	Ein-Ausgang, Datenbit 0
38	R/-W	Ausgang, Read/-Write
39	02	Taktausgang Phase Two im folgenden 02 genannt
40	-RES	Eingang, Reset

Wie viele andere Prozessoren hat also auch der 6510 einen 8-Bit-Daten- und einen 16-Bit-Adressbus. Somit kann der 6510 einen Speicherbereich von 64 K direkt adressieren.

Die Signale 0IN und 02 sind die Taktsignale des Systems, sozusagen der Herzschlag des Rechners. Das Signal 0IN wird vom VIC erzeugt und hat eine Frequenz von ungefähr 985 KHz. Aus diesem Signal wird im Prozessor das Signal 02 erzeugt. 02 ist für das Zusammenspiel von Prozessor und Peripherie sehr wichtig, es stellt den Bezugstakt für alle Operationen des Prozessors dar.

Das Signal -RES wird benutzt, um den Prozessor und andere ICs in einen definierten Anfangszustand zu versetzen. Dieser Reset findet im Einschaltmoment statt.

Schauen wir uns diesen Einschaltmoment einmal etwas näher an.

Das -RES-Signal wird vom IC U20 erzeugt.

Dies IC, ein NE556 enthält 2 identische Timer-Baustufen.

Mit diesen Timern kann man durch einfache externe Beschaltung Oszillator- oder Impulsgeberbaustufen aufbauen. In unserem Fall ist das IC als Impulsgeber geschaltet.

Mit dem Anlegen der Betriebsspannung wird der Kondensator C105 über den Widerstand R50 aufgeladen. Gleichzeitig wird der Kondensator C24 über den Widerstand R34 aufgeladen. Wenn nun nach einiger Zeit (einigen 10 Millisekunden) die Spannung am C105 den Wert von 1.6 Volt ( $1/3$  der Betriebsspannung) übersteigt, wird der eigentliche Impuls gestartet. Der Kondensator C24 wird über den Anschluss 13 schlagartig entladen. Gleichzeitig wird der Pin 9, der Ausgang des Timers, auf 5 Volt gelegt. Danach wird C24 über den Widerstand R34 wieder aufgeladen. Aber jetzt wird die Spannung durch den Eingang Pin 12 überwacht.

In dem Moment, wo die Spannung  $2/3$  der Betriebsspannung (ca. 3.3 Volt) übersteigt, wird der Ausgang wieder Low. Dieser Zeitpunkt ist nach etwa .5 Sekunden erreicht. Der am Ausgang Pin 9 des Timers befindliche Inverter macht aus diesem positiven Impuls einen Negativen. An seinem Ausgang steht das eigentliche -Res-Signal zur Verfügung.

Im Moment des Wechsels von High nach Low startet der Prozessor seine Arbeit. Als erstes holt er von den Adressen \$FFFC und \$FFFD (genannt Reset-Vektor) die Adresse des nächsten zu verarbeitenden Befehls. Auf dieser Adresse beginnt nun das eigentliche Betriebssystem.

Der Pin mit der Bezeichnung R/-W signalisiert, ob der Prozessor einen Lese- oder einen Schreibzugriff vornimmt. Wenn diese Leitung High ist, liest der Prozessor Daten aus Ram, Rom oder Interfacechips. Bei einem Low auf dieser Leitung schreibt der Prozessor, d.h. er speichert Daten im jeweils adressierten Baustein. Dieses Schreiben ist natürlich nur dann sinnvoll, wenn der adressierte Baustein diese Daten auch speichern kann. Auf ein Rom zu schreiben ist wenig sinnvoll, da die Daten des Rom schon bei der Herstellung festgelegt werden und nicht veränderbar sind.

Der Pin mit der Bezeichnung -NMI (Non Maskable Interrupt = nicht maskierbare oder ausblendbare Unterbrechung) gestattet eine Unterbrechung eines gerade laufenden Programms. Nicht Maskierbar bedeutet, das der Interrupt immer zugelassen ist. Er ist durch Software nicht auszuschließen.

Wann immer dieser Anschluß nach Masse gezogen wird, wird mit der Beendigung des gerade abgehandelten Maschinensprachbefehls das laufende Programm verlassen. Der Prozessor holt vom NMI-Vektor (\$FFFA und \$FFFB) die Adresse der Interrupt-Routine, und verzweigt auf diese.

Der NMI kann im CBM 64 durch drei verschiedene Ereignisse ausgelöst werden.

Der erste Fall ist das Drücken der RESTORE-Taste.

Wird diese Taste gedrückt, dann erzeugt der zweite Timer des U20 einen geeigneten Impuls.

Das Drücken der Taste entlädt den Kondensator C38 schlagartig. Über den Widerstand R35 wird der Kondensator wieder aufgeladen, auch wenn die RESTORE-Taste noch gedrückt ist. Sobald die Spannung am Pin 6 des U20 1.6 Volt

übersteigt, wird der eigentliche NMI-Impuls gestartet. Der Ausgang des Timers Pin 5 wird High, am Ausgang des Inverters U6 erscheint ein Low-Pegel, der Kondensator C23 wird über den Pin 1 von U20 entladen und beginnt sich über R33 wieder aufzuladen.

Nach ca. 18 Microsekunden ist der C23 auf 2/3 der Betriebsspannung aufgeladen und der Ausgang Pin 5 wird wieder Low, der -NMI-Eingang des Prozessors ist wieder High.

Der zweite Fall wird durch die CIA U2 erzeugt. Der Pin 21 dieses ICs kann beim Eintreffen bestimmter Ereignisse einen Low-Pegel annehmen.

Die Erzeugung dieses -NMI wird im Abschnitt über die CIAs behandelt.

Der dritte Fall ist das Kurzschließen des Anschluß D der Cartridge Expansion. Hier können externe Bausteine einen Interrupt auslösen.

Dem -NMI ähnlich ist der -IRQ (Interrupt ReQuest). Als wesentliche Unterschiede zum -NMI sind zum einen der Interruptvektor des -IRQ zu sehen. Dieser Vektor liegt auf den Adressen \$FFFE und \$FFFF. Des weiteren ist dieser Interrupt softwaremäßig ausschaltbar.

Wenn im Prozessorstatusregister das I-Flag (Bit 2) gesetzt ist, werden alle auftretenden Interrupts ignoriert.

Ein weiterer Unterschied zum -NMI ist die Tatsache, daß der -IRQ nicht flankengesteuert ist. Der Interrupt muss also mindestens so lange anliegen, bis der Prozessor diesen Anschluss prüft.

Erzeugt wird der -IRQ auch wieder auf drei verschiedene Arten.

Die CIA U1 erzeugt an seinem Pin 21 genau wie die CIA U2 einen Low-Pegel beim Erreichen bestimmter programmierbarer Zustände. Dieser Low-Pegel erzeugt einen -IRQ am Prozessor.

Die zweite Möglichkeit der Interrupterzeugung ist der VIC. Am Pin 8 des VIC erscheint genau wie bei den CIAs beim Erreichen bestimmter, vorher durch Programmierung festgelegter Ereignisse ein Lowpegel und damit der -IRQ.

Die dritte Möglichkeit der -IRQ-Erzeugung besteht in Kurzschließen des Anschlusses 4 des Cartridge Expansion Steckers (CN6). Somit haben auch externe Schaltungen die Möglichkeit der -IRQ-Generierung.

Der RDY-Pin zeigt dem Prozessor, ob die auf dem Datenbus liegenden Informationen gültig sind oder nicht.

Immer wenn dieser Pin Low ist, wird dem Prozessor signalisiert, daß er die Daten noch nicht übernehmen kann. Der Prozessor geht dann in einen sogenannten Wartezustand und stellt seine Aktivitäten ein. Er prüft nur mit jedem Taktimpuls, ob der RDY-Pin wieder High ist.

In älteren Prozessorsystemen wurde diese Möglichkeit genutzt, um langsame Speicher- und Peripheriebausteine am Prozessor anzuschließen. Im CBM 64 wird dies Signal vom VIC genutzt.

Normalerweise geschieht der Zugriff des VIC auf das Ram nur in den vom Prozessor nicht genutzten Taktlücken (Ø2 = Low).

Bei bestimmten Operationen des VIC, z.B. Darstellung der Sprites, benötigt der VIC mehr Zeit als in den Taktlücken zur

Verfügung steht. Dann erzeugt der VIC am Anschluß BA (Bus Available) ein Low, welches über das AND-Gatter U27 an den RDY-Eingang des Prozessors geführt wird, worauf der Prozessor den Bus dem VIC für die benötigte Zeit zur Verfügung stellt.

AEC ist ebenfalls ein in der Grundkonfiguration vom VIC erzeugtes Signal.

Immer wenn der VIC den Bus belegt, wird dieser Anschluß 0. Dieses Low-Signal wird an den AEC-Pin des Prozessors geführt und bewirkt, daß der Prozessor seine Busleitungen in einen hochohmigen, den sogenannten Tri-State-Zustand versetzt. In der Praxis wirkt das, als ob der Prozessor gar nicht in seinem IC-Sockel säße. Solange AEC Low ist, bleibt dieser Zustand erhalten und andere ICs, z.B ein externer Prozessor oder der VIC können den Systembus belegen.

Der im Prozessor integrierte Port belegt die Pins 24 bis 29. Im CBM 64 werden verschiedene Aufgaben von diesem Port übernommen. Im Einzelnen sind das die folgenden Funktionen: Das Portbit 0 trägt die Bezeichnung -LOWRAM. Dieses Bit schaltet im Adressbereich \$A000 bis \$BFFF zwischen RAM und ROM, d.h. bei Low-Pegel ist in diesem Adressbereich RAM eingeschaltet.

Portbit 1 mit der Bezeichnung -HIRAM übernimmt die selbe Funktion im Adressbereich von \$E000 bis \$FFFF.

Portbit 2 mit der Bezeichnung -CHAREN selektiert, wenn es einen Low-Pegel hat, das Character-ROM.

Character-ROM und der sogenannte IO-Bereich belegen den selben Adressbereich von \$D000 bis \$DFFF. Über -CHAREN wird also entschieden, ob das Character-ROM oder die den gleichen Adressbereich benutzenden IO- oder Peripherie-Bausteine VIC, SID oder CIAs selektiert sind.

Die drei verbleibenden Bits sind für den Betrieb der Datasette reserviert.

Die Schreibdaten für die Datasette werden vom Portbit 3 geliefert. Dieser Prozessorpin wird direkt auf die Anschlüsse E und 5 des Cassettenports geführt.

Portbit 4 (Cass Sense) überprüft, ob an der Datasette die Play-Taste gedrückt ist. Dieses Bit liegt direkt an den Anschlüssen F und 6 des Cassettenports.

Die Motorsteuerung des Recorders wird von Bit 5 übernommen. Die Funktion der Motorsteuerung wurde schon im Kapitel Stromversorgung erläutert.

## Adressdekodierung

Da der 6510 nur einen Adressraum von 64 K verwalten kann, dieser aber schon von den 64 K RAM belegt wird, muß eine zusätzliche Logik die Verwaltung der sich teilweise überlappenden Speicherbereiche übernehmen. Diese Verwaltung ist in der Hauptsache in einem speziellen IC integriert, dem sogenannten Adress-Manager. Im Schaltplan trägt dies IC, ein FPLA (Field Programmable Logic Array), die Bezeichnung U17. Erst durch die Programmierung hat dies IC seine besonderen Logikeigenschaften erhalten und ersetzt eine große Anzahl verschiedener Gatter, die nötig wären, wollte man die Funktion des AM mit herkömmlichen ICs nachbilden.

Die Pin-Belegung dieses 28-poligen ICs sieht folgendermaßen aus:

Pin	Bez.	Funktion
1	FE	Nicht benutzt
2	I7	Eingang, A13 vom 6510 Pin 20
3	I6	Eingang, A14 vom 6510 Pin 22
4	I5	Eingang, A15 vom 6510 Pin 23
5	I4	Eingang, -VA14 vom CIA 2 Port A Bit 0 Pin 2
6	I3	Eingang, -CHAREN vom 6510-Port Bit 2 Pin 27
7	I2	Eingang, -HIRAM vom 6510-Port Bit 1 Pin 28
8	I1	Eingang, -LOWRAM vom 6510-Port Bit 0 Pin 29
9	I0	Eingang, -CAS vom VIC Pin 19
10	F7	Ausgang, -ROMH zum Expansion Slot Pin 8
11	F6	Ausgang, -ROML zum Expansion Slot Pin 11
12	F5	Ausgang, -I/O zum Decoder U15 Pin 1
13	F4	Ausgang, GR/-W zum Farbram U6 Pin 10
14	GND	Betriebsspannung Masse
15	F3	Ausgang, -CHAROM zum Character-Rom U5 Pin 20
16	F2	Ausgang, -KERNAL zum Kernel-Rom U4 Pin 20
17	F1	Ausgang, -BASIC zum Basic-Rom U3 Pin 20
18	F0	Ausgang, -CABRAM zu den Rams Pin 15
19	-OE	Eingang, Output Enable an Masse
20	I15	Eingang, -VA12 vom VIC Pin 28
21	I14	Eingang, -VA13 vom VIC Pin 29
22	I13	Eingang, -GAME vom Expansion Slot Pin 8
23	I12	Eingang, -EXROM vom Expansion Slot Pin 9
24	I11	Eingang, R/-W vom 6510 Pin 30
25	I10	Eingang, -AEC vom VIC Pin 16
26	I9	Eingang, BA vom VIC Pin 12
27	I8	Eingang, A12 vom 6510 Pin 19
28	Vcc	Betriebsspannung +5 V

Was bewirken jetzt die verschiedenen Eingangssignale an den Ausgängen des AM? Bei 16 Eingangsleitungen sind ja immerhin 65536 verschiedene Eingangskombinationen möglich. Da der AM jedoch nur 8 Ausgänge besitzt, ist schon ersichtlich, das jeweils mehrere Eingangskombinationen eine bestimmte Ausgangskombination bewirken.

Aber auch unter den 256 möglichen Ausgangskombinationen sind nur wenige für den Computer wirklich sinnvoll. Diese

sinnvollen Kombinationen sind zur besseren Übersichtlichkeit in der Tabelle auf S. 53 und den anschließenden Seiten mit den Speicherzuordnungsplänen dargestellt.

Übrigens, wenn jede mögliche Eingangskombination und die dazugehörige Ausgangskombination eine Zeile einer Seite belegen würde, dann hätte eine vollständige Liste bei dem von uns verwendeten Druckformat immerhin einen Umfang von fast 1093 Seiten.

## Der Videocontroller 6569

Die beiden wichtigsten Peripherie-Geräte eines Computers sind Eingabe- und Ausgabe-Einheiten, da sie die Möglichkeit schaffen, mit dem Computer in Verbindung zu treten.

Die Ausgabe-Einheit des CBM 64 ist in der Regel der Fernseher oder ein Monitor.

Der VIC stellt im CBM 64 alle für den Betrieb eines Fernsehers oder Monitors benötigten Signale zur Verfügung. Dies sind die Sync- und Helligkeitsimpulse und die für Farbdarstellung benötigten Farbwerte.

Zusätzlich übernimmt der VIC aber noch andere Aufgaben. So erzeugt er den von der CPU benötigten Takt, übernimmt den bei den verwendeten dynamischen RAMs notwendigen Refresh und liefert Steuersignale für den Betrieb der dynamischen RAMs.

Diese Funktionen sind alle in einem 40-poligen Gehäuse untergebracht. Die Belegung der Pins ist in der folgenden Tabelle dargestellt.

Pin	Bez.	
1	D6	Prozessordatenbus
bis		
7	D0	Prozessordatenbus
8	-IRQ	Ausgang, Interrupt Request
9	-LP	Eingang, Light Pen
10	-CS	Eingang, Chip Select
11	R/-W	Read/-Write
12	BA	Bus Available
13	VDD	Betriebsspannung +12 Volt
14	COLOR	Ausgang, Farbinformation
15	SYNC	Ausgang, Zeilen- und Bildsynchronisationsimpulse
16	AEC	Ausgang, Address Enable Control
17	00OUT	Ausgang, Systemtakt
18	-RAS	Ausgang, Row Address Select
19	-CAS	Ausgang, Column Address Select
20	GND	Betriebsspannung Masse
21	00COLOR	Eingang, Farbfrequenz
22	00IN	Eingang, Dotfrequenz
23	A11	Prozessoradressbus
24	A0/AB	gemultiplixter (Video-) Ram-Adressbus
bis		
29	A5/A13	gemultiplixter (Video-) Ram-Adressbus
30	A6	(Video-) Ram-Adressbus
31	A7	(Video-) Ram-Adressbus
32	A8	Prozessoradressbus
bis		
34	A10	Prozessoradressbus
35	D11	Datenbus Farbram
bis		
38	D8	Datenbus Farbram
39	D7	Prozessordatenbus
40	VCC	Betriebsspannung +5 Volt

Wenn Sie sich die verschiedenen Pin-Bezeichnungen am VIC anschauen, dann treffen Sie auf einige bekannte Bezeichnungen. So sind BA, AEC, Ø2, und R/-W schon beim Prozessor erläutert worden. Völlig neu sind z.B. die Signale -CS, -RAS, -CAS und die Datenleitungen DB - D11. Auch der gemultiplixte Adressbus ist neu hinzugekommen, da am Prozessor ja alle Adresssignale getrennt an einzelnen Pins zu Verfügung standen.

Doch kommen wir zuerst zu den verschiedenen Taktsignalen. Das den ganzen Zeitablauf im Rechner bestimmende Signal ist der Dot-Clock. Dieses Signal hat in Ihrem CBM 64 eine Frequenz von ca. 7.05 MHz. Im VIC befindet sich eine Stufe, die diese Frequenz durch 8 teilt. Damit erhalten wir eine neue Frequenz von ca. 980 KHz. Diese Frequenz steht am Pin 17 als Systemtakt ØOut zur Verfügung. Aus dem Dot-Clock werden weiterhin die Signale zur Synchronisation des Bildes auf dem Fernseher gewonnen. Der Dot-Clock selbst bestimmt die Zeit, mit der die einzelnen Punkte, aus denen alle Zeichen dargestellt werden, auf dem Bildschirm erscheinen.

Die Frequenz des Signals ØCOLOR beträgt in Ihrem CBM 64 17.734472 MHz. Das ist die Frequenz mit der der Quarz Y1 schwingt. Sie wird zur Erzeugung der Farbinformation benötigt.

Diese Frequenzen beziehen sich alle auf den Normalfall, d.h. der Rechner ist für den Betrieb mit einem PAL-System-Fernseher ausgestattet.

Immer wenn der Prozessor auf die Register des VIC zugreifen will, muß der VIC adressiert werden. Dazu muß als wichtigstes die Leitung mit der Bezeichnung -CS auf Low gehen. Erst dann kann der Prozessor über die auf dem Adressbus liegende Adresse das gewünschte Register ansprechen. Wie wird nun aber die Leitung -CS Low.

Da der VIC im sogenannten IO-Bereich (\$D000 bis \$DFFF) die Adressen von \$D000 bis \$D3FF belegt, erzeugt der AM bei einem Zugriff auf diesen Adressbereich einen Low-Pegel an seinem Pin 12 (-I/O-Signal). Dieser Low-Pegel gelangt an den Dekoder U15 Pin 1. Damit ist der Dekoder freigegeben und in Abhängigkeit von den Adressleitungen A10 und A11 an den Pin 2 und 3 wird der entsprechende Ausgang des Dekoders Low. Wenn man die Basisadresse und Endadresse des VIC einmal binär darstellt, so erhält man das folgende Bitmuster:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0=\$D000
1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1=\$D3FF

Man sieht sofort, das die Adressbits A10 und A11 in diesem



Adressbereich Low bleiben. Damit ist der Ausgang Y0 des Dekoders auf Low, der VIC ist adressiert. Erst bei der nächsten Adresse \$D400 wird A10 High. Damit wird Y0 High, Y1 des Dekoders wird Low und nun ist der SID adressiert.

Der VIC kann nur einen Adressraum von 16 K adressieren, er hat nur die Adressbits A0 bis A13. Außerdem liegen die Adressleitungen nicht wie beim Prozessor einzeln an den Pins an, sondern sind gemultiplext. Der Pin 24 ist also nicht nur Adressbit 0 sondern auch Adressbit 8. Wie kann das funktionieren?

Die Antwort ist ganz einfach. Der Anschluss ist erst das eine Adressbit, danach das andere. Um jetzt zu einem bestimmten Zeitpunkt sagen zu können, welche Bedeutung der Anschluß hat, werden Hilfssignale benötigt.

Diese Hilfssignale heißen -CAS und -RAS. Sie werden unter anderem auch zur Steuerung der dynamischen RAM-Bausteine benötigt, da diese auch einen gemultiplexten Adressbus aufweisen.

Der zeitliche Ablauf des Speicherzugriffs sieht folgendermaßen aus.

Die Signale -CAS und -RAS sind high. Jetzt wird zuerst das niederwertige Adressbyte auf den Bus gelegt. Nach kurzer Zeit wird das Signal -RAS Low. Damit wird das Adressbyte in die RAMs übernommen und gespeichert. Jetzt ändert sich die Businformation. Aus A0 wird A8, aus A1 wird A9 usw. Wiederrum nach kurzer Zeit wird jetzt das Signal -CAS Low. Diese abfallende Flanke wird auf den AM gegeben und erzeugt am Ausgang -CASRAM eine zeitlich geringfügig verzögerte abfallende Flanke. Mit dieser verzögerten Flanke wird nun das High-Byte in die RAMs übernommen.

Jetzt liegt die vollständige Adresse vor und die Daten erscheinen auf dem Datenbus. Diese Vorgänge sind in Timing-Diagramm auf der nächsten Seite noch einmal dargestellt.

Die Schnittstelle zwischen RAM und VIC.

Da wie schon gesagt der VIC nur die Adressbits A0 bis A13 erzeugt, müssen die für die Adressierung der ganzen 64 k Ram fehlenden Bits zusätzlich erzeugt werden.

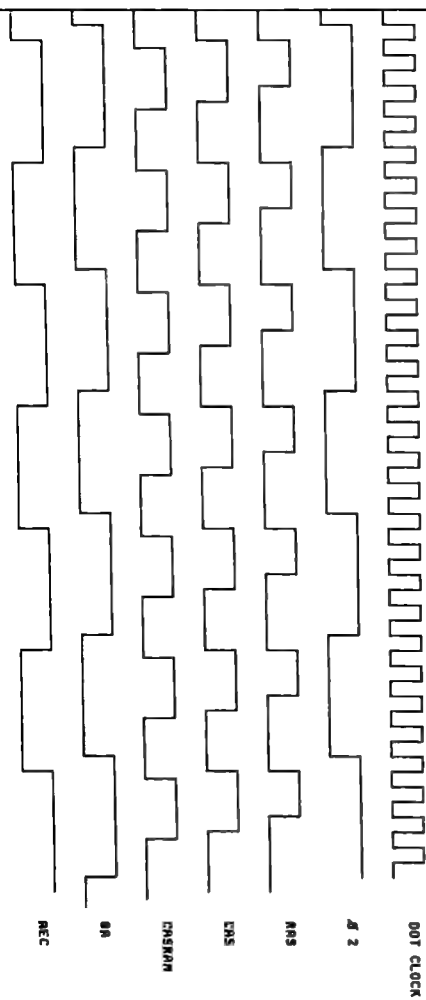
Dazu wird der Port A der CIA 2 herangezogen. Die Portbits 0 und 1 stellen die Adressbits 14 und 15 dar. Um diese Signale in den Multiplex-Vorgang einzubeziehen, werden sie über das IC U14 geschaltet.

Im U14 sind vier invertierende 2 zu 1 Multiplexer integriert. So ein Multiplexer ist in seiner Funktion wohl am einfachsten als Wechselschalter zu sehen. Wahlweise einer von zwei Eingängen wird auf den zugehörigen Ausgang geführt.

Im Detail funktioniert die Sache so:

Geschaltet werden die Multiplexer durch das Signal am Eingang S. Liegt an S ein Low, dann sind die Eingänge mit der Bezeichnung A auf den Ausgang durchgeschaltet, liegt S auf

# Phasenlage der Bussteuer Signale



High-Pegel, dann sind die B-Eingänge durchgeschaltet. Die Adressbits A6 und A7 vom VIC liegen am Multiplexer, und zwar A6 an den Eingängen 13 und 14 und A7 an den Eingängen 10 und 11. Wenn jetzt mittels des S-Signals zwischen den Eingängen hin- und hergeschaltet wird, so ist an den Ausgängen keine Änderung festzustellen, da die Adressbits auf beide Eingänge geführt sind. Nur die Polarität der Signale ist an den Ausgängen durch die Inverterwirkung der Multiplexer vertauscht. Diese invertierten Adresssignale werden auf die B-Eingänge der beiden anderen Multiplexer geführt, und zwar -A6 an den Pin 3 und -A7 an den Pin 6. Die A-Eingänge werden mit den genannten Portbits der CIA 2 versorgt, Portbit 0 als -VA14 an Pin 2, Portbit 1 als -VA15 an Pin 5.

Da der S-Eingang von -CAS gesteuert wird, liegt am Ausgang Pin 4 bei -CAS High das nochmal invertierte Adressbit A6, bei -CAS Low das Adressbit A14. Am Ausgang Pin 7 wird entsprechend zwischen -A7 und -VA15 geschaltet. Durch die Invertierung des Multiplexers erscheint dies Signal als A7 oder A15.

Der Pin 15 von U14 ist mit dem Signal AEC verbunden. Er trägt die Bezeichnung -OE, Output Enable.

Immer wenn AEC High ist, werden die Ausgänge des U14 abgeschaltet oder in den sogenannten Tri-State-Zustand versetzt. Dies ist wichtig, da bei AEC High der Prozessor den Bus belegt und seine Adressen über die Multiplexer U13 und U25 auf diesen Bus legt. Nur wenn AEC Low ist, kann der VIC ja den Bus belegen, dann sind die Ausgänge des U14 freigegeben.

16 Farben mit vier Bits, das Farb-Ram.

Sollen alle 512 möglichen Zeichen auch noch in 16 verschiedenen Farben dargestellt werden, dann sind vier weitere Datenbits erforderlich. Es sind dies die vier Pins 35 bis 38 am VIC. An diesen Pins ist das Color-Ram U6 mit seinen Datenleitungen angeschlossen. Dies IC ist ein statisches RAM mit 4096 Speicherplätzen. In jedem Speicherplatz kann ein Bit gespeichert werden. Jeweils 4 Speicherplätze werden durch eine Adresse angesprochen.

Die Adressierung geschieht zuerst wieder durch das Signal -CS am U6. Wenn dieser Anschluss Low ist, wird das RAM selektiert, die Datenleitungen verlassen den Tri-State-Zustand.

Erzeugt wird das -CS-Signal auf zwei verschiedene Arten vom AND-Gatter U27. So seltsam es auch klingen mag, dieses AND-Gatter wird in der Schaltung als ein OR-Gatter betrieben. Ein AND-Gatter legt den Ausgang dann auf High, wenn alle Eingänge auch High sind. Wenn man jetzt die Logik ein wenig umdreht, kann man auch sagen, wenn der eine ODER der ander Eingang Low ist, dann ist der Ausgang auch Low. Diese Betriebsart wird im CBM 64 angewendet.

Das Color-Ram belegt den Adressbereich von \$D800 bis \$DBFF. Wenn das Signal AEC High ist, belegt der Prozessor den Bus. Damit ist der eine Eingang des AND-Gatters High. Wenn der

Prozessor nicht auf das Color-Ram zugreift, ist der Ausgang -COLOR des Dekoders U15 auch High. Damit ist der -CS-Eingang des Color-Ram auf High, das Farbram ist nicht selektiert. Wenn der Prozessor auf das Farbram zugreifen will, legt er die entsprechende Speicheradresse auf den Datenbus. Die Dekodierung läuft entsprechend wie die des VIC ab. Nur ist jetzt mit Sicherheit das Adressbit A11 gesetzt. Damit wird der -COLOR-Ausgang des Dekoders U15 Low. Jetzt ist ein Eingang des AND-Gatters Low und entsprechend der Ausgang auch. Damit ist das Farbram selektiert.

Da AEC zu diesem Zeitpunkt High ist, sind die vier Analogschalter im IC U16 geschlossen, die Datenleitungen des Farbram sind mit den vier niederwertigen Datenleitungen des Prozessors verbunden. Damit kann nun das Farbram beschrieben und gelesen werden.

Wenn AEC Low wird und der VIC den Bus übernimmt, dann werden die Analogschalter geöffnet. Gleichzeitig wird der Ausgang des AND-Gatters U27 Pin 8 Low, das Farbram ist selektiert, diesmal vom VIC. Da der VIC aber nur mit den Adressleitungen A8 bis A11 mit dem VIC verbunden ist, müssen die Adressbits A0 bis A7 anders gewonnen werden. Diese Aufgabe übernimmt das IC U26. Dies TTL-IC mit der Bezeichnung 74LS373 enthält 8 Latches oder Zwischenspeicher. Die Eingänge dieses ICs sind mit dem gemultiplexten Adressbus verbunden. Eingespeichert werden die Daten, wenn das Signal -RAS Low wird. Das ist der Zeitpunkt, wenn das niederwertige Adressbyte auf dem Bus liegt. Die Ausgänge von U16 sind mit dem niederwertigen Adressbyte des Prozessorbusses verbunden und liefern die Adressinformationen, wenn der Prozessor im Tri-State-Zustand ist.

Auf diese Weise kann der VIC das Farbram adressieren.

Auch der Zwischenspeicher ist mit dem Signal AEC verbunden. Am Pin 1 des U16 bewirkt es im High-Zustand, daß die Ausgänge hochohmig werden um den Prozessor nicht zu stören.

Wenn man sich diese Vorgänge genauer anschaut, ergibt sich eine interessante Frage. Wieso hat der VIC zusätzlich zum gemultiplexten Adressbus A0 bis A13 noch die vier Adressleitungen A8 bis A11 an den Pins 23, 32, 33 und 34 ?

Die Antwort ist relativ simpel. Der VIC muss zu jeder Bildschirmspeicheradresse im Bereich von \$0400 bis \$07FF gleichzeitig die entsprechende Farbspeicherzelle im Adressbereich \$D800 bis \$DBFF ansprechen. Dieser gleichzeitige Zugriff auf zwei verschiedene Speicherplätze erfordert einen zweiten, vom normalen Adressbus unabhängigen Bus. Dieser Bus wird durch die 4 separaten Adressbits realisiert.

## Der Character-Generator.

Um auf dem Bildschirm ein einzelnes Zeichen darzustellen, werden  $8 \times 8$  Punkte benötigt. Da ein Byte 8 Bits enthält, sind zur Darstellung also 8 Bytes nötig. Das hört sich natürlich etwas verwirrend an, im Bildspeicher wird ja pro Zeichen nur ein Byte belegt. An einem Beispiel wollen wir uns das Ganze einmal etwas genauer anschauen. Damit werden die Zusammenhänge etwas klarer.

Betrachten wir einmal den Buchstaben A.

Löschen sie den Bildschirm (Gleichzeitiges Drücken der 'SHIFT'- und 'CLR/HOME'-Taste) und dann drücken Sie die Taste A. Jetzt schauen Sie sich das Ergebnis einmal ganz genau an. Abhängig von der Qualität Ihres Fernsehers oder Monitors werden Sie mehr oder weniger das genaue Punktemuster dieses Zeichens erkennen können.

Ganz genau sieht das A so aus:

	1	2	3	4	5	6	7	8
A.	.	.	.	*	*	.	.	.
B.	.	.	*	.	.	*	.	.
C.	.	*	.	.	.	.	*	.
D.	.	*	*	*	*	*	*	.
E.	.	*	.	.	.	.	*	.
F.	.	*	.	.	.	.	*	.
G.	.	*	.	.	.	.	*	.
H.	.	.	.	.	.	.	.	.

In den Reihen A bis H repräsentiert jedes '\*' ein High des entsprechenden Bits, ein '.' entspricht einem Low. Jede Reihe enthält somit 8 Bits, ein Byte. Bei 8 Reihen kommen wir damit nach allen Regeln der Mathematik auf 8 Bytes pro dargestelltes Zeichen. Diese 8 Bytes sind im Charakter-Rom gespeichert.

Wie wir sehen können, enthält der im Bildschirmspeicher stehende Code nicht direkt das Zeichen.

Tatsächlich ist dieser Bildschirmcode eine Adresse, und zwar die Grundadresse des Zeichens im Charakter-Rom. Um aber alle 8 Adressen eines Zeichens im CH-ROM zu adressieren sind 3 weitere Adressleitungen nötig. Diese Aufgabe übernehmen jetzt die drei Leitungen A8 bis A10 des separaten Adress-Busses. A11, die noch freie Leitung dieses Adressbusses erfüllt eine besondere Aufgabe.

Der Bildschirmspeicher kann ein Byte gleich 8 Bit speichern. Damit sind 256 verschiedene Zeichen im CH-ROM adressierbar. Der 64 kann aber 512 verschiedene Zeichen darstellen.

Die Organisation der 512 Zeichen im 64 sieht etwa folgendermaßen aus.

Der CBM 64 verfügt über 2 mögliche Zeichensätze. Der erste, nach dem Einschalten benutzte Zeichensatz erlaubt die Darstellung von Großbuchstaben und Grafikzeichen. Dafür werden 128 Zeichen benötigt. Zusätzlich sind alle Zeichen auch revers darstellbar, macht zusammen 256 mögliche Zeichen.

Die zweite Darstellungsart wird durch gleichzeitiges Drücken der Commodore- und der 'SHIFT'-Taste eingeschaltet. Dieser Zeichensatz erlaubt die Darstellung der Klein- und Großbuchstaben. Auch das sind wieder 128 verschiedene Zeichen, die zusätzlich noch revers dargestellt werden können, insgesamt also 512 verschiedene Zeichen. Dieses Umschalten zwischen den beiden Zeichensätzen wird mit dem noch freien Adressbit A11 vorgenommen.

#### Der Prozessor und das Ram.

Bisher haben wir uns nur mit dem Fall beschäftigt, das der VIC auf die 64 K Arbeitsspeicher zugreifen will. Es fehlt noch die Beschreibung der Vorgänge bei einem Zugriff des Prozessors auf dieses Ram.

Die Leszugriffe des Prozessors sind den Zugriffen des VIC sehr ähnlich. In beiden Fällen liegt das Signal R/-W (lesen bei High, schreiben bei Low) auf High.

Zuerst darum die Leszugriffe.

Wie bei der Beschreibung der Ram-VIC-Schnittstelle erläutert, benötigt das Ram einen gemultiplexten Adressbus. Diese Forderung der Rams kann der Prozessor aber nicht erfüllen. Darum ist ein Multiplexen mit zusätzlichen ICs notwendig.

Diese Multiplexer sind die ICs U13 und U25, zwei 74LS257. Diese ICs arbeiten nach dem selben Prinzip wie das U14 (beschrieben im Abschnitt RAM und VIC). Der Unterschied zu U14 besteht darin, daß diese Multiplexer die Ausgangssignale nicht invertieren.

An den Eingängen der beiden Multiplexer-ICs liegt der komplette Prozessoradressbus A0 bis A15. Dabei sind die Eingänge so geschaltet, das mit dem Select-Signal jeweils zwischen A0 und A8, A1 und A9 u.s.w. umgeschaltet wird. Die Adressierung der Rams läßt sich wieder in drei Phasen zerlegen.

In der ersten Phase liegt am Select-Eingang der Multiplexer ein High. Damit ist das niederwertige Adress-Byte auf die Rams geschaltet.

Mit der abfallenden Flanke des -RAS-Signals wird dies Byte in die RAMs übernommen.

Kurze Zeit später wird auch das -CAS-Signal Low. Damit schalten die Multiplexer um, der jeweils zweite Eingang der Multiplexer wird auf die entsprechenden Ausgänge geschaltet und das höherwertige Adressbyte liegt an den RAMs.

Über den AM wird das Signal -CAS wieder etwas verzögert. Der Ausgang -CASRAM übernimmt auch hier die eigentliche Funktion des Signals -CAS.

Mit der abfallenden Flanke vom -CASRAM wird nun das High-Byte der Adresse in den RAMs gespeichert.

Jetzt wird in den RAMs die adressierte Speicherzelle angesprochen und die Daten erscheinen auf dem Datenbus.

Die Schreibzugriffe des Prozessors unterscheiden sich von den Leszyklen durch einen wesentlichen Umstand.

Bei einem Schreibzugriff wird der Prozessor-Pin R/-W Low,

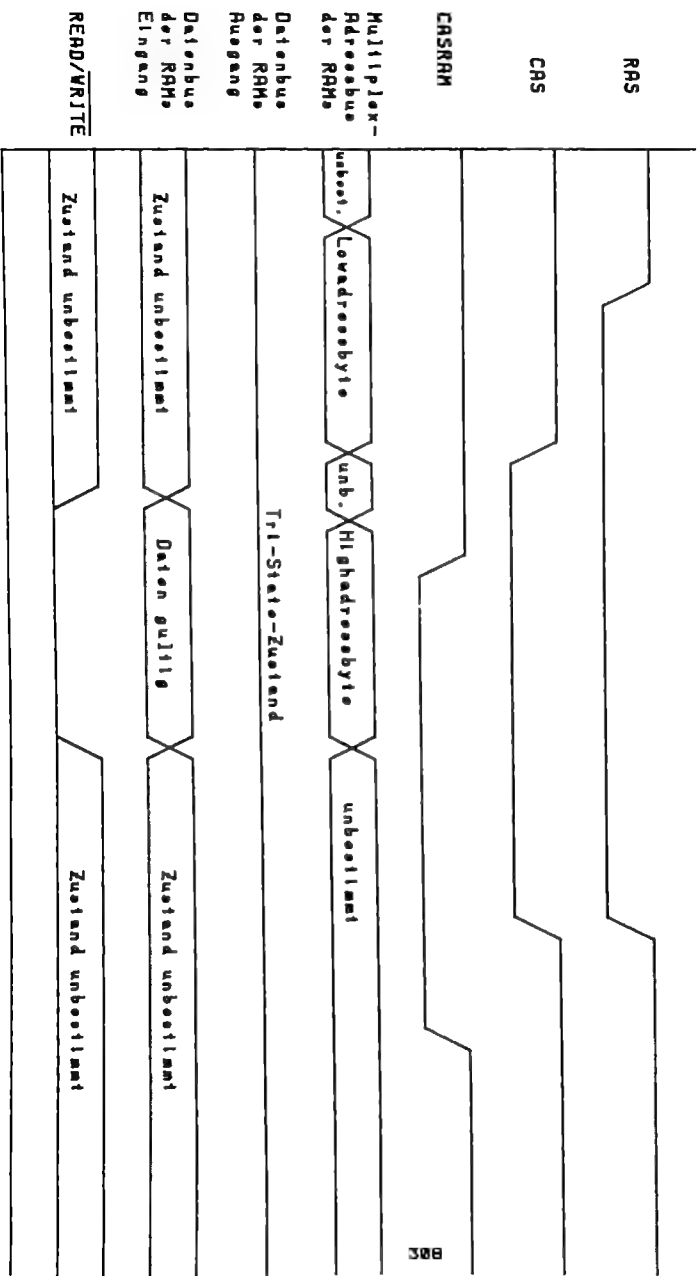
nachdem der Prozessor die Adresse der entsprechenden Speicherzelle auf den Adressbus gelegt hat. Damit ist dem RAM signalisiert, daß das auf dem Datenbus liegende Byte in dieser Speicherzelle gespeichert werden soll. Die verwendeten RAM-Bausteine stellen an dies R/-W-Signal eine bestimmte Bedingung.

Das Signal R/-W darf erst dann Low werden, nachdem -RAS Low geworden ist, -CASRAM aber noch High ist. R/-W muss also zwischen den beiden abfallenden Flanken von -RAS und -CASRAM Low werden.

Der zeitliche Verlauf der -RAS-, -CAS- und -CASRAM-Signale ist mit denen bei Lesezugriffen identisch.

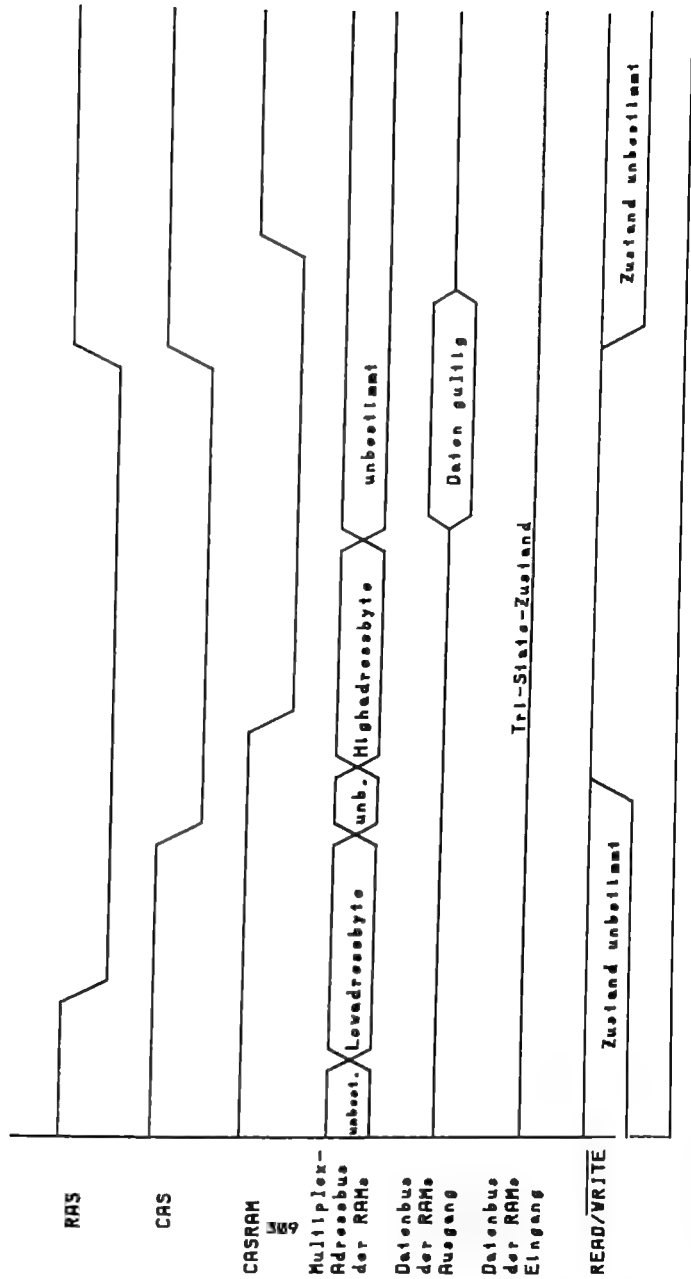
Zur besseren Verdeutlichung dieser Vorgänge sind die Signale für Schreib- und Lesezugriffe in den Bildern auf der nächsten Seite dargestellt.

# Schreibzugriff auf das dyn. RAM





# Lesezugriff auf das dyn. RAM



Der SID 6581, ein Syntesizer mit 28 Beinen und mehr.

Dieses IC ist genau wie der VIC ein Paradebeispiel für die Möglichkeiten der Halbleiterindustrie. Durch dies IC erhält der CBM 64 seine fantastischen Klangmöglichkeiten. Vor wenigen Jahren hätte allein ein Syntesizer mit diesen in einem IC integrierten Möglichkeiten die ganze Leiterplatte des 64 für sich in Anspruch genommen.

Die 28 Pins des 6581 haben die folgenden Bezeichnungen:

Pin	Bezeichnung	
1	CAP1A	Externer Kondensator für Frequenzfilter
2	CAP1B	Wie Pin 1
3	CAP2A	Wie Pin 1
4	CAP2B	Wie Pin 1
5	-RES	Eingang, Reset-Signal
6	Ø2	Eingang, Taktsignal
7	R/-W	Eingang, Read/-Write
8	-CS	Eingang, Chip Select
9	A0	Eingang, Adressbit 0
bis		
13	A4	Eingang, Adressbit 4
14	GND	Betriebsspannung Masse
15	D0	Datenbit 0, bidirektional
bis		
22	D7	Datenbit 7, bidirektional
23	POTY	Eingang, AD-Wandler 2
24	POTX	Eingang, AD-Wandler 1
25	Vcc	Betriebsspannung +5V
26	EXT IN	Eingang, externe Signalquelle
27	AUDIO OUT	Ausgang Syntesizer
28	Vdd	Betriebsspannung +12V

Die meisten Signale sind bereits aus den vorherigen Kapiteln bekannt.

Nicht vorgekommen sind bisher die Bezeichnungen der ersten vier Pins, CAP1A bis CAP2B. Wie man im Schaltbild sehen kann, sind an diesen Anschlüssen die 2 Kondensatoren C10 und C11 angeschlossen. Diese Kondensatoren werden für die im Chip U18, dem SID, integrierten Frequenzfilter benötigt.

Ein Filter ist eine uns allen bekannte Einrichtung. Nehmen wir zum Beispiel mal das Kaffee-Filter. Die Aufgabe dieses Filters ist es, bestimmte Anteile (nämlich das Wasser und die löslichen Stoffe des Kaffee-Pulvers) durchzulassen und andere Anteile (in unserem Beispiel die Reste des Kaffee-Pulvers) zurückzuhalten.

Genau so arbeitet auch ein elektronisches Frequenzfilter. Bestimmte Frequenzen werden durchgelassen, andere werden zurückgehalten.

Es gibt insgesamt vier mögliche Arten von Frequenzfiltern, den Tiefpass, den Hochpass, den Bandpass und den Sperrpass. Ein Tiefpass läßt nur tiefe Frequenzen bis zu einer bestimmten höchsten Frequenz passieren. Diese Funktion ist an jeder Stereoanlage in Form des Bass-Reglers zu finden. Mit diesem Regler läßt sich diese höchste, durchzulassende

Frequenz, die sogenannte Grenzfrequenz einstellen.

Ein Hochpass zeigt genau das umgekehrte Verhalten, ab einer bestimmten niedrigsten Frequenz läßt er alle höheren Frequenzen durch. Das ist an der Stereoanlage der Treble- oder Höhenregler.

Bleiben noch Bandpass und Sperrpass. Auch diese haben genau entgegengesetzte Funktionen.

Ein Bandpass ist eine Mischung aus Tief- und Hochpass. Ab einer bestimmten Frequenz werden höhere Frequenzen durchgelassen, dies aber nur bis zu einer höchsten Frequenz. Darüberliegende Frequenzen werden wieder gesperrt.

Das Sperrfilter sperrt in einem bestimmten Frequenzbereich alle Frequenzen. Diese Funktion ist an manchen guten Stereoanlagen als BrummfILTER vorhanden. Damit wird in diesem Fall nur eine bestimmte Frequenz, die 50 Hertz der Netzfrequenz herausgefiltert.

Alle diese Filter lassen sich im SID programmieren.

Der Pin 5, der Reseteingang von U18, wird benötigt, um das IC in einen definierten Zustand zu bringen. Wie bereits beschrieben, liegt an diesem Anschluss nach dem Einschalten für ca. 0.5 Sekunden ein Low-Pegel. Damit werden alle Register im 6581 gelöscht.

Ohne diesen Reset würden die Register nach dem Einschalten zufällige Werte haben, die Folge wäre ein zufälliges Signal am Audioausgang, der angeschlossene Fernseher oder Verstärker würde nur 'Krach' machen.

Aus der Frequenz des Signals 02 werden alle Tonfrequenzen des SID erzeugt. Gleichzeitig stellt 02 natürlich wie bei allen anderen Peripherie-Bausteinen den Bezugstakt für die Schreib- und Lesezugriffe des Prozessors dar.

Ob die im SID enthaltenen Register beschrieben oder gelesen werden, hängt wieder von der Leitung R/-W ab. Bei einem High werden die Register gelesen, bei einem Low wird in die Register geschrieben. Voraussetzung ist natürlich, das der SID auch korrekt adressiert ist.

Der Adressbereich des 6581 liegt von \$D400 bis \$D7FF. Dieser Adressbereich wird wie beim VIC durch den AM und die Dekoder im IC U15 dekodiert. Sobald der Prozessor eine Adresse in diesem Bereich auf den Bus legt und das Signal -CHAREN High ist, wird der Ausgang Pin 5 des 74LS139 Low, damit auch der -CS-Eingang des SID.

Um nun auch die einzelnen Register im SID zu adressieren, werden die 5 Adressleitungen A0 bis A4 benötigt.

Sind diese Adressbits alle Low und der SID mit -CS selektiert, kann das Register 0 beschrieben oder gelesen werden. Ist nur das Adressbit A0 High, ist Register 1 selektiert usw.

Auf diese Weise lassen sich alle 29 Register ansprechen.

Die Datenleitungen D0 bis D7 an den Pins 15 bis 22 sind mit dem Prozessordatenbus verbunden. So lange -CS High ist, befinden sich die Datenleitungen des SID im Tri-State-

Zustand. Wenn der -CS Low wird, entscheidet R/-W, ob die Datenleitungen als Eingang (beim Schreiben der Register) oder als Ausgang (entsprechend beim Lesen) fungieren.

Die Anschlüsse POTX und POTY stellen die Eingänge der AD-Wandler dar.

Bis jetzt ist die Bezeichnung AD noch nicht erläutert worden. Das wollen wir schnell nachholen.

AD-Wandler ist die Abkürzung für Analog-Digital-Wandler. Ein digitaler Wert kennt bekanntlich nur zwei Zustände, entweder High oder Low, im CBM 64 und vielen anderen Digital- und Computerschaltungen durch eine Spannung von +5V als High und 0 Volt als Low signalisiert.

Ein analoges Signal ist da nicht so festgelegt, es kann jeden beliebigen Wert dazwischen, darüber und darunter annehmen.

Nun ist es aber oft wünschenswert, einen solchen analogen Wert in einen Computer eingeben zu können, um ihn zu verarbeiten. Diese Möglichkeit der Eingabe analoger Werte ist im CBM 64 eingebaut.

Hauptsächlich genutzt werden die AD-Wandler in Verbindung mit den Paddles, das sind Drehregler, ähnlich den Reglern an Radiogeräten.

Ein solcher Regler enthält einen veränderbaren Widerstand, Potentiometer oder kurz Poti genannt. Der Widerstandswert des Potis ändert sich mit dem Drehen. Der minimale Widerstand der in den Paddles eingebauten Potis beträgt ca. 100 Ohm, der Maximalwert ca. 500 KOhm. Dazwischen kann der Widerstand theoretisch jeden beliebigen Wert annehmen.

Der AD-Wandler erzeugt aus diesem Widerstandswert ein digitales Signal, in unserem Fall wird ein 8-Bit-Signal erzeugt. Dieses Byte kann aus einem der SID-Register gelesen werden.

Die eigentliche AD-Wandlung geschieht mit dem eingestellten Widerstandswert und den Kondensatoren C48 und C93.

Diese Kondensatoren werden für 0.25 Millisekunden über die Potis aufgeladen. Wenn die Spannung an den Kondensatoren größer wird, als die im SID erzeugte Vergleichsspannung, wird ein Zähler im SID angehalten, der Zählerstand ist das Maß für den eingestellten Widerstand. Je größer der Widerstand des Potis ist, um so langsamer wird der Kondensator aufgeladen, und die Spannung am Kondensator erreicht die Höhe der Referenzspannung später. Damit kann der Zähler länger laufen, der Zählerwert wird größer.

Ist der Widerstandswert zu hoch (ca. 200 KOhm), dann erreicht die Spannung am Eingang des AD-Wandlers in der Meßzeit nicht die Referenzspannung. Der Zähler läuft dann bis zu seinem Endwert, im AD-Register steht der Wert 255.

Wenn der Widerstand aber zu klein wird (ca. 200 Ohm), ist der Kondensator so schnell aufgeladen, daß der Zähler sofort gestoppt wird. Damit steht im Register ein Wert von 0.

Nach Ablauf der Meßzeit von 0.25 Millisekunden werden die Kondensatoren schlagartig über den entsprechenden AD-Eingang entladen. Jetzt wird der Zähler auf 0 gesetzt und nach weiteren 0.25 Millisekunden startet dann ein neuer Meßzyklus. Somit benötigt ein vollständiger Zyklus 0.5 Millisekunden, in einer Sekunde werden 2000 mal die aktuellen Widerstandswerte

gemessen und stehen zur Verfügung.

Um eine Beschädigung der AD-Eingänge zu vermeiden, sollte der Widerstand nicht kleiner als 100 Ohm werden. Sonst werden die bei der Entladung der Kondensatoren auftretenden Ströme zu groß, und die Entladestufe am Eingang wird zerstört.

Die zwei Eingänge POTX oder POTY liegen aber nicht direkt an einer der verschiedenen Buchsen des 64. Die beiden Eingänge liegen an den Pins 2, 3, 9 und 10 des IC U28. Dies IC, ein CMOS-Baustein mit der Bezeichnung 4066, enthält vier sogenannte Analogschalter. Dies IC wird benötigt, da an den 64 zwei Paddlepaare, insgesamt also vier Potis, angeschlossen werden können.

So ein Analogschalter arbeitet vergleichbar einem Relais. Wenn am Steuereingang eine Spannung anliegt, wird der Analogeingang auf den Ausgang durchgeschaltet, der Schalter ist geschlossen. Liegt der Steuereingang auf Masse, dann ist der Ausgang vom Eingang gesperrt, der Schalter ist geöffnet. Die Analogeingänge sind mit den Controllports CN8 und CN9 verbunden. An diesen Controllports sind die Kontakte 5 und 9 für den Anschluß der Paddles vorgesehen.

Die Steuereingänge sind die Pins 5, 6, 12 und 13. Der Pin 13 kontrolliert den Schalter 1 zwischen den Anschlüssen 1 und 2, Pin 5 den Schalter 2 zwischen 4 und 3, Pin 6 den Schalter 3 zwischen 8 und 9 und Pin 12 Schalter 4 zwischen 11 und 10. Jeweils zwei dieser Eingänge sind zusammengeschaltet, Pin 13 und 5 und Pin 6 und 12.

Diese jeweils verbundenen Eingänge liegen an den beiden Pins 8 und 9 der CIA U1. Über diese Leitungen kann man auswählen, welche Potis an den Eingängen des AD-Wandlers liegen. Sind die Pins 8 und 9 der CIA U1 Low, dann liegt kein Poti an den Wandlern. Ist Pin 8 High, dann sind die Analogschalter 3 und 4 geschlossen, die am Controllport 1 angeschlossenen Paddles werden an die Anschlüsse POTX und POTY gelegt. Ist dagegen Pin 9 der CIA High, dann sind Analogschalter 1 und 2 geschlossen, die Paddles an CN8 liegen an den Eingängen der AD-Wandler.

Bleiben noch die Anschlüsse EXT IN und AUDIO OUT am 6581.

AUDIO OUT ist der NF-Ausgang des Synthesizers. Hier stehen die im Synthesizer erzeugten Töne und Geräusche zur Verfügung. Bei maximaler Lautstärke hat das Ausgangssignal eine Größe von 2V<sub>ss</sub>.

Der Transistor Q8 ist als Emitterfolger an den Ausgang geschaltet. Dadurch, daß das Signal am Emitter des Transistors über dem Widerstand R38 abgenommen wird, hat der Transistor keine Spannungsverstärkung. Das Signal am Ausgang Pin 3 des 8-poligen Video-Audio-Buchse CN5 hat somit auch eine Höhe von 2V<sub>ss</sub>.

An diesen Ausgang kann man direkt einen kleinen 8-Ohm-Lautsprecher anschliessen. Allerdings ist die Lautstärke sehr gering. Um eine vernünftige Wiedergabe zu erreichen, geben Sie das Signal am besten auf eine Stereoanlage oder ein gutes Kofferradio. Oder Sie benutzen den im Fernseher eingebauten Lautsprecher und das mit dem Bild Übertragene Tonsignal.

EXT IN gibt die Möglichkeit auch externe Signale in den Synthesizer einzuspeisen und zu beeinflussen. Externe Signale können beispielsweise Mikrofonsignale sein, die mit einem kleinen Verstärker verstärkt worden sind. Auch eine Gitarre oder eine Orgel kann nach entsprechender Verstärkung das Eingangssignal liefern, oder aber auch ein zweiter SID, also ein zweiter CBM 64. Damit hätte man dann noch wesentlich mehr Möglichkeiten der Klanggestaltung. Die einzige an das Eingangssignal gestellte Forderung lautet, daß das Signal nicht größer als 3V<sub>eff</sub> sein darf. Dieser Eingang ist über den Kondensator C12 mit dem Kontakt 5 der 8-poligen Audio-Video-Buchse CN5 verbunden.

## Die CIAs 6526

Diese beiden Bausteine mit den Bezeichnungen U1 und U2 erfüllen eine Vielzahl von verschiedenen Aufgaben im CBM 64. Die Tastatur- und Joystickabfrage, der serielle Datenbus zu Floppy und Drucker, eine optionale RS-232 Serialschnittstelle, die im vorigen Kapitel beschriebene Ein- und Umschaltung der Analogeingänge, Teile der Datensettensteuerung und die Erzeugung der schon erwähnten Hilfsadressbits A14 und A15 für den VIC, all diese Aufgaben werden von den zwei CIAs erledigt.

Ein solcher CIA (Complex Interface Adapter) 6526 mit seinen 40 Anschlüssen enthält 16 einzeln programmierbare Ein-Ausgabeleitungen, zwei Intervalltimer, eine Echtzeitzuhr mit programmierbarer Alarmzeit und ein 8-Bit-Schieberegister für serielle Ein-Ausgabe.

Doch betrachten wir uns zunächst wieder die einzelnen Anschlüsse des ICs:

### Pin Bezeichnung

1	GND	Betriebsspannung Masse
2	PA0	Ein-Ausgabeport A Bit 0
bis		
9	PA7	Ein-Ausgabeport A Bit 7
10	PB0	Ein-Ausgabeport B Bit 0
bis		
17	PB7	Ein-Ausgabeport B Bit 7
18	-PC	Ausgang, Port Control
19	TOD	Eingang, Time Of Day
20	Vcc	Betriebsspannung +5V
21	-IRQ	Ausgang, Interrupt Request
22	R/-W	Eingang, Read/-Write
23	-CS	Eingang, -Chip Select
24	-FLAG	Eingang, wie Port Control
25	02	Eingang, Systemtakt
26	D7	Prozessordatenbus
bis		
33	D0	Prozessordatenbus
34	-RES	Eingang, Resetsignal
35	RS3	Eingang, Register Select
bis		
38	RS0	Eingang, Register Select
39	SP	Serial Port, bidirektional
40	CNT	Count, bidirektional

Die Leitungen mit den Bezeichnungen PA0 bis PA7 und PB0 bis PB7 sind die 16 bidirektionalen Ein-Ausgabeleitungen. Je nach Programmierung stellen diese Leitungen Ein- oder Ausgänge dar. Damit besteht die Möglichkeit, Daten als 8- oder auch als 16-Bitwort zu lesen oder auszugeben.

Die CIA 1 legt diese 16 IO-Leitungen auf den Stecker, mit dem die Tastatur angeschlossen ist. Die Tastatur ist als eine

Matrix von jeweils 8 mal 8 Leitungen aufgebaut. Wenn man die Tasten des 64 zählt, dann kommt man auf 66 Tasten. In einer 8 mal 8 Matrix lassen sich aber nur 64 Tasten abfragen. Die Lösung dieses Problems sind die Tasten RESTORE und SHIFT LOCK.

Die RESTORE-Taste ist nicht in der Matrix enthalten, wie schon beschrieben schaltet diese Taste den Eingang von U20 nach Masse und erzeugt einen NMI.

Die Taste SHIFT LOCK ist einfach zu einer SHIFT-Taste parallel geschaltet und benötigt somit keinen eigenen Platz in der Matrix.

Die exakte Position jeder Taste in der Matrix entnehmen Sie bitte dem Bild auf der nächsten Seite.

Vor der Erklärung der Tastaturabfrage eine kurze Erklärung. Wenn bei den CIAs ein Portbit als Eingang geschaltet ist, dieser Eingang aber nicht belegt ist, dann erkennt die CIA an diesem Anschluss ein High-Signal.

Aber nun die eigentliche Abfrage. Vergleichen Sie dazu bitte auch das Bild auf der nächsten Seite.

Die Leitungen PA0 bis PA7 sind als Ausgang geschaltet, die Leitungen PB0 bis PB7 als Eingänge.

Wenn das Betriebssystem die Tastatur abfragen will, werden die Anschlüsse von Port A für einen kurzen Moment Low. Wenn auf der Tastatur zum Beispiel der Buchstabe 'H' gedrückt ist, wird in diesem Moment auch Bit 5 des Ports B Low. Damit bemerkt der Rechner, das eine der Tasten F3, S, F, H, K, I, = oder die Commodore-Taste gedrückt ist.

Welche dieser 8 Tasten gedrückt ist, kann zu diesem Zeitpunkt noch nicht erkannt werden.

Ist aber ein Tastendruck erkannt, werden nacheinander die Ausgänge von Port A kurz Low, jeweils einer zur Zeit. Nach jedem Wechsel der Ausgänge wird an den Eingängen von Port B geprüft, ob ein Eingang Low ist.

Das ist in unserem Beispiel dann der Fall, wenn das Bit 3 des Ports A Low wird. Damit liegt die Position der gedrückten Taste innerhalb der Matrix fest. Die genaue Anordnung der Tasten und der Anschluß an die CIA ist aus der Zeichnung auf der Seite 310 zu sehen.

Die Joystickabfrage wird auch von CIA 1 erledigt.

Ein Joystick enthält nichts anderes als 5 Schalter. Vier diese Schalter sind für die vier Richtungen, der fünfte Schalter ist der Feuerknopf. Diese Schalter liegen aber nicht in einer Matrix sondern legen jeweils ein Portbit an Masse.

Der Joystick 1 liegt auf den Portbits 0 bis 4 von Port B, Joystick 2 auf den Portbits 0 bis 4 von Port A.

Zusätzlich übernimmt Port A der CIA 1 mit den Bits 6 und 7 die Umschaltung der Paddles. Diese Umschaltung wurde bereits im Kapitel über die AD-Wandler erläutert.



Der Pin 19 der CIAs trägt die Bezeichnung TOD. TOD (Time Of Day) ist ein Eingang, der die in den CIAs integrierte Echtzeituhr mit den Taktsignalen versorgt. An diese Eingänge wird das mit dem Gatter U27 erzeugte 50 Hertz-Rechtecksignal geführt. In den CIAs wird diese Frequenz durch 5 geteilt, damit steht eine Frequenz von 10 Hertz zur Verfügung. 10 Hertz bedeuten 10 Impulse pro Sekunde, jede zehnte Sekunde ein Impuls, die kleinste mit den CIAs meßbare Zeiteinheit.

Der Pin 21 trägt die Bezeichnung -IRQ. Wie auch beim VIC wird dieser Anschluss zur Erzeugung von Interrupts genutzt. Bei der CIA 1 ist er mit dem -IRQ-Eingang des Prozessors verbunden.

Der Anschluss wird dann Low, wenn bestimmte programmierbare Ereignisse in der CIA auftreten.

R/-W, der Pin 22 steuert wieder die Art des Datentransfers. Wenn die Register der CIA 1 und 2 gelesen sollen, ist dieser Anschluss High, beim Schreiben in diese Register ist er Low.

Wie alle Peripherie-Bausteine im CBM 64 sind auch die CIAs Memory-Mapped. Das bedeutet, das Sie diese Bausteine genau wie Speicherplätze im RAM ansprechen können.

Bei jeder Aktivität des Prozessors in dem Adressbereich \$DC00 bis \$DFFF wird, wenn das Signal -CHAREN High ist, der Ausgang Pin 7 des Dekoders U15 Low. Damit ist gleichzeitig der Enable-Eingang des zweiten im U15 enthaltenen Dekoders auf Low, der Dekoder 2 ist freigegeben. An den Eingängen Pin 13 und 14 dieses zweiten Dekoders liegen die Adressbits A8 und A9. Mit diesen Adressbits wird der durch den Dekoder 1 dekodierte Adressbereich in vier kleinere Adressbereiche unterteilt. Jeder der auf diese Weise gewonnenen Adressbereiche ist 256 Bytes groß.

Die ersten 256 Bytes belegen den Adressbereich \$DC00 bis \$DCFF (Pin 12 von U15) und selektieren die erste CIA, der zweite Bereich liegt von \$DD00 bis \$DFFF (Pin 11) und selektiert CIA 2.

Die beiden freien dekodierten Adressbereiche von \$DE00 bis \$DEFF und von \$DF00 bis \$DFFF werden als IO1 und IO2 auf die Anschlüsse 7 und 10 der Cartridge Expansion geführt. Der IO1 wird zum Einschalten der CP/M-Cartridge benötigt, IO2 wird von keiner uns bekannten CBM-Erweiterung benutzt. Damit ist dieser Adressbereich vorzüglich für eigene Projekte geeignet.

Wenn auf diese Weise eine der beiden CIAs selektiert ist, bestimmen die Leitungen RS0 bis RS3, welches interne Register in der CIA angesprochen werden soll. Dazu sind diese Eingänge mit den vier niederwertigen Adressbits des Prozessorbusses verbunden.

Die Datenleitungen D0 bis D7 (Anschlüsse 33 bis 26 an den CIAs) sind mit dem Prozessordatenbus verbunden. Über diese Leitungen werden Daten ausgelesen respektive in die Register hineingeschrieben.

Der Anschluss mit der Bezeichnung -FLAG der CIA 1 erfüllt

eine Doppelfunktion.

Zum einen ist er mit den Anschlüssen D und 4 des Cassettenports CN3 verbunden. Über diese Leitung werden die von der Datensette gelieferten Wiedergabesignale eingelesen. Zum zweiten ist der -FLAG-Eingang mit Pin 1 des Serialbus verbunden. Dieser Pin 1 von CN4 trägt die Bezeichnung -SRQIN. Über diesen Service ReQuest Eingang können Peripherie-Geräte dem 64 melden, wenn sie Daten für den Rechner zur Verfügung haben. Diese Funktion wird aber von keinen Geräten der Firma Commodore genutzt.

Dieser -FLAG-Eingang ist flankengetriggert. Jeder Wechsel von High nach Low wird im CIA signalisiert und kann einen Interrupt an Pin 21 erzeugen.

Der Eingang mit der Bezeichnung -PC kann als Hand Shake Signal für den Port B oder für Port A und Port B zusammen genutzt werden.

Wenn sie in einer Anwendung den Port B als Eingabeport benutzen, dann können Sie mit einem negativen Impuls an -PC die Daten in den Computer übergeben. Damit können 8-Bit-Daten übernommen werden.

Je nach Programmierung werden mit abfallender Flanke des Pegels an -PC nur die Daten an Port B oder an beiden Ports eingelesen.

Bei der CIA 1 ist dieser Eingang nicht benutzt.

Pin 39 mit der Bezeichnung SP (Serial Port) kann je nach Programmierung wahlweise der Eingang oder der Ausgang des Schieberegisters im CIA sein.

Mit einem Schieberegister läßt sich eine serielle Schnittstelle ohne großen Aufwand realisieren.

Dieser Anschluss der CIA 1 liegt auf dem Anschluß 5 des User-Ports CN2, und wird von einer aufgesteckten RS-232-Schnittstelle benutzt. Ohne RS-232 können Sie diesen Anschluss für eigene Anwendungen verwenden.

Der Anschluß CNT der CIA 1 liegt auch auf dem User-Port. Dort belegt er den Pin 4. Über diesen Anschluß kann der Schieberegistertakt ausgegeben oder auch eingegeben werden. Zusätzlich kann dieser Anschluß als Takteingang für die eingebauten Zähler in der CIA programmiert werden. Auch dieser Anschluss wird nur bei aufgesteckter RS-232 benutzt.

Der Port A der CIA 2 ist aufgeteilt, d.h er erfüllt mehrere Funktionen.

Bit 0 des Ports A ist das Videohilfsadressbit VA 14, Bit 1 des Ports ist VA 15. PA2 ist das einzige Bit des Ports A, das innerhalb des 64 keine Funktion erfüllt. Es liegt 'nur' am User-Port auf dem Anschluss M. Die verbleibenden Portbits 3 bis 7 werden für den seriellen IEC-Bus verwendet. Dabei ist das Bit 3 der Ausgang zur Erzeugung des -ATN-Signals. Dieser Anschluß wird an den Pin 1 des IC U8 geführt.

Das IC U8 enthält 6 sogenannte Buffer, Signalverstärker die das Signal nicht invertieren. Außerdem sind die Buffer mit Open Collector-Ausgängen versehen, sie benötigen am Ausgang einen Arbeitswiderstand nach +5V.

Der Ausgang des Buffers für das -ATN-Signal Pin 2 ist verbunden mit Pin 3 des seriellen Busses, gleichzeitig aber auch mit dem Userport Pin 9. Damit kann das Portbit auch für Ihre Schaltungen am Userport genutzt werden, allerdings nur als Ausgang.

Die Portbits 4 und 5 werden auch über das IC U8 gepuffert. Portbit 4 ist das Signal CLK OUT, Bit 5 das Signal DATA OUT. Die Ausgänge der beiden Buffer sind mit den Pins 4 und 5 des Serialbusses verbunden als Signale CLK an Pin 4 und DATA an Pin 5. Im Ruhezustand sind diese Ausgänge High.

Gleichzeitig liegen aber auch die Portbits 6 und 7 an den Ausgängen der Buffer. Der Grund dafür ist die Tatsache, das sowohl CLK als auch DATA bidirektionale Signale sind, sie werden nicht nur im 64 erzeugt, sondern auch in einer angeschlossenen Floppy oder einem Drucker. Die Portbits 6 und 7 sind also entsprechend als Eingänge programmiert und ein externes Gerät kann diese beiden Signale nach Masse ziehen.

Alle 8 Leitungen des Ports B liegen nur auf dem User-Port, und stehen damit Ihnen, dem User oder Benutzer zur Verfügung. Da diese Leitungen sowohl als Eingänge als auch als Ausgänge zu benutzen sind, können sowohl Daten aus dem Computer ausgegeben werden als auch Daten in den 64 eingegeben werden. Detaillierte Angaben über die Programmierung und Verwendung des User-Ports finden Sie im Kapitel 1.6 dieses Buches.

Der Kontakt B des User-Ports ist mit dem Anschluß -PC (Pin 18) der CIA 2 verbunden. Wie schon bei der CIA 1 erläutert, signalisiert eine negative Flanke an diesem Eingang die Gültigkeit der Daten an Port B für den Fall, daß der Port B als Eingang programmiert ist. Damit können Sie also dem 64 anzeigen, das die am User-Port anliegenden Daten in den Rechner übernommen werden können.

-FLAG der CIA 2 (Pin 24) liegt am User-Port-Kontakt B. Für den Fall der Programmierung des Ports B als Ausgangsport kann dieser Anschluß so programmiert werden, daß er die Gültigkeit der Daten an den Portleitungen anzeigt. Er hat also genau die umgekehrte Funktion des -PC-Eingangs, da er für die Datenrichtung aus dem Port heraus an die angeschlossene Peripherie zuständig ist.

Der -IRQ-Ausgang der CIA 2 wird ebenso wie bei der CIA 1 als Interrupt erzeugender Anschluß verwendet. Ein in der CIA 2 erzeugter Interrupt löst aber am Prozessor einen -NMI, durch Software nicht ausblendbaren Interrupt aus.

Die Adressbereichsdekodierung geschieht wie auch bei der CIA 1 durch die Dekoder im IC U15. Die CIA 2 belegt aber den Adressbereich von \$DD00 bis \$DDFF. Das an Pin 23 benötigte -CS-Signal wird am Pin 11 des Dekoders erzeugt.

An beiden CIAs liegt noch an Pin 34 das System-Reset-Signal. Damit werden auch in diesen Peripherie-Bausteinen alle Register nach dem Einschalten in einen definierten

Anfangszustand versetzt.

Die Funktion der übrigen Signale an U2 braucht nicht erläutert werden. Diese Anschlüsse haben die selbe Funktion wie die entsprechenden Leitungen an der CIA 1.

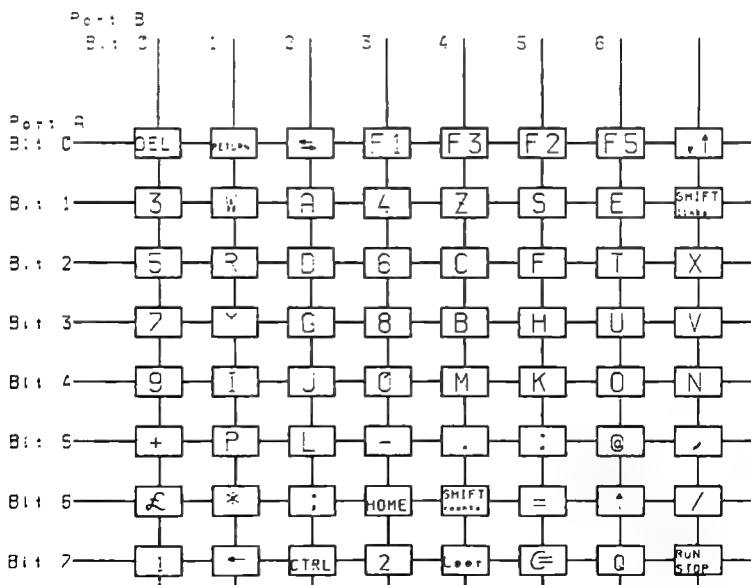
### Der Modulator.

Anders als beim VC-20 ist der Modulator des 64 bereits im Gerät eingebaut.

Leider ist kein Schaltplan des Modulators verfügbar. Trotzdem eine kurze Beschreibung der Vorgänge in diesem Teil.

Der Modulator besteht in der Hauptsache aus einem Oszillator, der auf einer Frequenz im UHF-Fernsehbereich schwingt. Diesem Oszillatorsignal werden die Signale SYNC+LUM und COLOR des VIC 6569 und das Audio-Signal des SID 6581 aufmoduliert. Das entstehende Signal ist an der Chinch-Buchse verfügbar und kann mit einem Koax-Kabel an den Antenneneingang eines Fernsehers geführt werden.

Die durch die Öffnungen im Modulator sichtbaren Abgleichselemente sollten Sie NIE verstellen. Der Abgleich des Modulators wurde im Herstellerwerk vorgenommen und ist mit Sicherheit optimal.



# Liste der verwendeten Halbleiter.

Für die Spezialisten unter Ihnen hier noch eine Aufstellung der im CBM 64 verwendeten ICs mit Herstellerangaben. Alle ICs können Sie im Data Becker Chip Shop erhalten. Damit haben Sie die Möglichkeit, Ihr Gerät selbst zu reparieren. Allerdings sei an dieser Stelle DRINGEND davor gewarnt, ohne detaillierte Kenntnisse der Computerhardware oder einen entsprechenden Satz an Meßgeräten eine Reparatur zu versuchen. In den meisten Fällen ist die Reparatur nach einem solchen misslungenen Versuch teurer als die Reparatur des ursprünglichen Fehlers.

Bez.	Typenbez.	Hersteller
U1	6526 CIA	Commodore MOS
U2	6526 CIA	Commodore MOS
U3	2364A BASIC	Commodore MOS
U4	2364A KERNAL	Commodore MOS
U5	2332A CHARACTER	Commodore MOS
U6	2114L-3 COLOR RAM	diverse Hersteller
	z.B. OKI	MSM 2114L-3
	FAIRCHILD	2114L-3
	HITACHI	HM2114L-3
	MOS	MP82114L-30
	MOTOROLA	MCM2114L-30
	NEC	uPD2114L-1
U7	6510 MPU	Commodore MOS
U8	7406	diverse Hersteller
U9	4164 RAM	diverse Hersteller
U10		
U11	z.B. NEC	uPD4164-2
U12	MOSTEK	MK4164-10
U21		
U22		
U23		
U24		
U13	SN74LS257	diverse Hersteller
U14	SN74LS270	diverse Hersteller
U15	SN74LS139	diverse Hersteller
U16	MC4066	diverse Hersteller
U17	82S100	Signetics, programmiert durch Commodore
U18	6581 SID	Commodore MOS
U19	6589 VIC	Commodore MOS
U20	556	diverse Hersteller
U25	SN74LS257	diverse Hersteller
U26	SN74LS373	diverse Hersteller
U27	SN74LS00	diverse Hersteller
U28	MC4066	diverse Hersteller
U29	SN74LS74	diverse Hersteller
U30	SN74LS193	diverse Hersteller
U31	SN74LS629	diverse Hersteller
U32	MC4044	Motorola
VR1	7812 12V Regler	diverse Hersteller
VR2	7805 5V Regler	diverse Hersteller

Nun hat auch COMMODORE seinen Henkelmann.

Der in letzter Zeit verstärkt zu beobachtende Trend nach einer kompakteren Bauweise hat auch vor COMMODORE nicht haltgemacht.

In einem noch als handlich zu bezeichnenden Gehäuse mit den Abmessungen 330xH14xT42cm ist so ziemlich alles untergebracht, was zu einem Rechner gehört, das Ganze mit einem praktischen Tragegriff versehen.

Im einzelnen sind im Gehäuse eingebaut:

- \* die modifizierte Rechnerplatine des C64 in zwei Teilen
- \* die modifizierte Platine der Floppy VC-1541
- \* ein Lautsprecher mit 8cm Durchmesser
- \* ein Monitor (wahlweise monochrom oder farbig)
- \* wahlweise ein oder zwei Diskettenlaufwerke
- \* eine superflache Tastatur (mittlere Höhe nur 3cm), die bei Nichtgebrauch die Frontseite des Gerätes verschließt

An Bedienungselementen außer der Tastatur sind vorhanden:

- Netzschalter auf der Rückseite
- auf der Vorderseite hinter einem Türchen Regler für Lautstärke, Kontrast, Helligkeit, Farbsättigung und Rot-Grünbalance

An extern zur Verfügung stehenden Anschlüssen sind zu nennen:

- zwei Anschlüsse für Joysticks
- der serielle IEC-Bus
- Monitoranschluß (nicht moduliert)
- Userport
- durch Federklappen geschützter Expansionsport auf der Oberseite des Gerätes
- kein Anschluß für Kassettengeräte

Im Inneren des Rechners geht es naturgemäß ziemlich eng zu. Die Platinen sind senkrecht gestellt und durch steckbare Anschlüsse miteinander verbunden.

Durch ausreichend dimensionierte Lüftungsschlitze ist eine gute Wärmeabfuhr gewährleistet, sodaß thermische Probleme trotz der hohen Packungsdichte nicht zu erwarten sind.

Der Bildschirm, obwohl mit 5" (ca. 13cm) nicht gerade der größte, liefert ein erstaunlich scharfes Bild mit klaren Farben, was wohl auch darauf zurückzuführen ist, daß das Videosignal nicht erst über einen Modulator geleitet wird, was ja bei Betrieb mit einem handelsüblichen Fernsehgerät nötig wäre.

Daraus folgt, daß Sie, falls ein größeres Bild gewünscht wird, nicht ohne weiteres die Möglichkeit haben, einen Fernseher anzuschließen.

Besonderes Lob verdient die Tastatur, die wegen ihrer

geringen Bauhöhe und mit den leichtgängigen Tasten ein ermüdungsfreies Arbeiten ermöglicht.

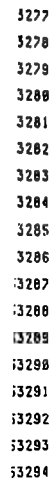
Zusammenfassend läßt sich sagen, daß es sich hier um ein Gerät handelt, welches alle wesentlichen Merkmale eines Personal-Computers in einer glücklichen Größe vereint, was sicherlich die oftmals leidgeprüften Ehegefährten von Computerfreaks geneigter macht, da weder der Küchentisch mit allerlei Gerätschaften blockiert werden muß, noch der (in allgemeinen nur einmal vorhandene) Fernseher dem häuslichen Gebrauch entzogen wird.

# CHIP : VIC (6567) Video-Controller

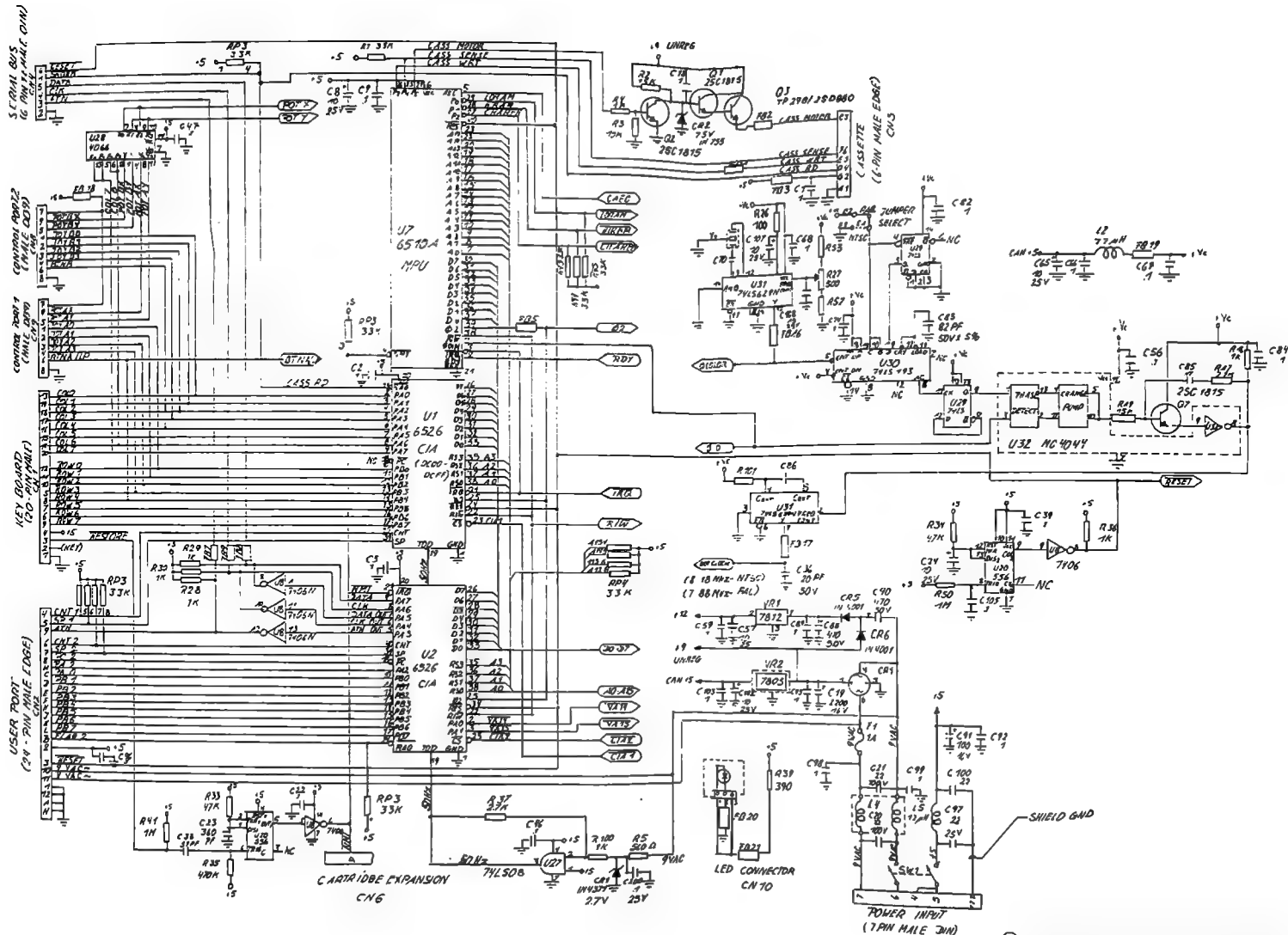
+0 00000	Sprite 0 X-Coordinate (0-255)								53248
+1 00001	Y-Coordinate Sprite 1								53249
+2 00002	X-Coordinate (0-255)								53250
+3 00003	Y-Coordinate Sprite 2								53251
+4 00004	X-Coordinate (0-255)								53252
+5 00005	Y-Coordinate Sprite 3								53253
+6 00006	X-Coordinate (0-255)								53254
+7 00007	Y-Coordinate Sprite 4								53255
+8 00008	X-Coordinate (0-255)								53256
+9 00009	Y-Coordinate Sprite 5								53257
+10 0000A	X-Coordinate (0-255)								53258
+11 0000B	Y-Coordinate Sprite 6								53259
+12 0000C	X-Coordinate (0-255)								53260
+13 0000D	Y-Coordinate Sprite 7								53261
+14 0000E	X-Coordinate (0-255)								53262
+15 0000F	Y-Coordinate Sprite X-Coordinate +256								53263
+16 00010	Sprite 7 Sprite 8 Sprite 9 Sprite 10 Sprite 11 Sprite 12 Sprite 13 Sprite 14								53264
+17 00011	Raster	Motorp.	Misc	Blid	20/24	Position			53265
+18 00012	RGB	Maskfab	Grnth	ang	Roller	TRCL2	TRCL3	TRCL4	53266
+19 00013	Raster								53267
+20 00014	RC2	RC3	RC4	RC5	RC6	RC7	RC8	RC9	53268
+21 00015	Light - Pen X-Coordinate Light - Pen Y-Coordinate Sprite on/off								53269
+22 00016	SE7	SE8	SE9	SE10	SE11	SE12	SE13	SE14	53270
+23 00017	Macht beuht Maskfab 40/38 Position								53271
+24 00018	M	P	P	M	M	TRCL2	TRCL3	TRCL4	53272
+25 00019	Sprite wargroun... T-Coordinate								53273
+26 0001A	SE17	SE18	SE19	SE20	SE21	SE22	SE23	SE24	53274
+27 0001B	Bildschirmspeicher Adr. (00000) Nicht								53275
+28 0001C	CE15	CE17	CE18	CE19	CE20	CE21	CE22	CE23	53276
	IA0	Nicht beuht			Light-P.	Sp.-Sp.	Sp.-Mte.	Raster	
	Flas	P	P	P	L.-Flas	Kell.	Kell.	L.-Flas	
	Nicht beuht				Hacke	Hacke	Hacke	Hacke	
	P	P	P	P	MLP1	MLP2	MLP3	MLP4	
	Sprite-Mintorgund Polarisant								
	SPF7	SPF8	SPF9	SPF10	SPF11	SPF12	SPF13	SPF14	
	Mehrfarbige Sprite								
	SMC2	SMC3	SMC4	SMC5	SMC6	SMC7	SMC8	SMC9	







© Commodore Büromaschinen GmbH 1983  
Jegliche Vervielfältigung oder Nachdruck  
ohne Erlaubnis des Urheberrechtinhabers ist  
untersagt und wird auf dem Rechtswege verfolgt.



3277  
3278  
3279  
3280  
3281  
3282  
3283  
3284  
3285  
3286  
3287  
3288  
3289  
3290  
3291  
3292  
3293  
3294



+29	00010	Spitta vargassarna X-Koordinata 9CXX7 I 9CAZ8 I 9CX19 I 9CXX4 I 9CXX3 I 9CAV8 I 9CXX1 I 9CXAP	53277
+30	0001E	Spitta-Spitta Kollstation 9PC7 I 9PCV I 9PC4 I 9PC4 I 9PC3 I 9PC2 I 9PC1 I 9PCP	53278
+31	0001F	Spitta-Mintergrund Kollstation 9PC7 I 9PCV I 9PC4 I 9PC4 I 9PC3 I 9PC2 I 9PC1 I 9PCP	53279
+32	00020	Rundfarbo (0-10) 9CXX1	53280
+33	00021	Mintergrundfarbo Nr. 0 (0-10) 9CXX2	53281
+34	00022	Mintergrundfarbo Nr. 1 (0-10) 9CXX1	53282
+35	00023	Mintergrundfarbo Nr. 2 (0-10) 9CXX1	53283
+36	00024	Mintergrundfarbo Nr. 3 (0-10) 9CXX2	53284
+37	00025	Coastguard Spitta-Farbo Nr. 0 i Rødtfarbo-Medue (0-10) 9CXX2	53285
+38	00026	Coastguard Spitta-Farbo Nr. 1 i Rødtfarbo-Medue (0-10) 9CXX1	53286
+39	00027	Farbo Spitta Nr. 0 (0-10) 9CXX4	53287
+40	00028	Farbo Spitta Nr. 1 (0-10) 9CXX4	53288
+41	00029	Farbo Spitta Nr. 2 (0-10) 9CXX4	53289
+42	0002A	Farbo Spitta Nr. 3 (0-10) 9CXX4	53290
+43	0002B	Farbo Spitta Nr. 4 (0-10) 9CXX4	53291
+44	0002C	Farbo Spitta Nr. 5 (0-10) 9CXX4	53292
+45	0002D	Farbo Spitta Nr. 6 (0-10) 9CXX4	53293
+46	0002E	Farbo Spitta Nr. 7 (0-10) 9CXX4	53294

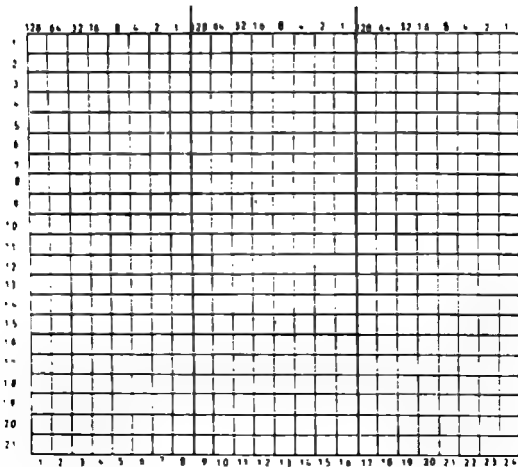
# CHIP : SID (6581) Sound Synthesizer

#D400	OSZILLATOR 1					54272
#D401	Frequenz, low Byte					54273
#D402	Frequenz, high Byte					54274
#D403	Pulsebreite, low byte					54275
#D404	Pulsebreite, high Byte					54276
#D405	Wellenform					54277
#D406	RAUSCH breitere Bereich, BREITE TEST RING SYNC KEY					54278
#D407	Anstiegszeit					54279
#D408	SUSTAIN					54280
#D409	Abklingzeit					54281
#D40A	OSZILLATOR 2					54282
#D40B	Frequenz, low Byte					54283
#D40C	Frequenz, high Byte					54284
#D40D	Pulsebreite, low byte					54285
#D40E	Pulsebreite, high Byte					54286
#D40F	Wellenform					54287
#D410	RAUSCH breitere Bereich, BREITE TEST RING SYNC KEY					54288
#D411	Anstiegszeit					54289
#D412	SUSTAIN					54290
#D413	Abklingzeit					54291
#D414	OSZILLATOR 3					54292
#D415	Frequenz, low Byte					54293
#D416	Frequenz, high Byte					54294
#D417	Pulsebreite, low Byte					54295
#D418	Pulsebreite, high Byte					54296
#D419	Wellenform					54297
#D41A	RAUSCH breitere Bereich, BREITE TEST RING SYNC KEY					54298
#D41B	Anstiegszeit					54299
#D41C	SUSTAIN					54300
#D41D	Abklingzeit					54301
#D41E	Filterfrequenz					54302
#D41F	Resonanz					54303
#D420	EXT OS3 OS2 OS1					54304
#D421	Filterart					54305
#D422	Lautstärke					54306
#D423	PADDLE 1					54307
#D424	PADDLE 2					54308
#D425	RAUSCHEN 3 (Zufallszahl)					54309
#D426	Hüllkurve 3					54310

CHIP : CIA (6526)

+0 BDCB0	PRB								56328
	PR7	PR8	PR9	PR1	PR3	PR2	PR1	PR8	
+1 BDCB1	PRB								56321
	PR7	PR8	PR9	PR1	PR3	PR2	PR1	PR8	
+2 BDCB2	BDRB								56322
	BPR7	BPR8	BPR9	BPR1	BPR3	BPR2	BPR1	BPR8	
+3 BDCB3	BDRB								56323
	BPR7	BPR8	BPR9	BPR1	BPR3	BPR2	BPR1	BPR8	
+4 BDCB4	TR LB								56324
	TR17	TR18	TR19	TR1	TR3	TR2	TR1	TR8	
+5 BDCB5	TR HI								56325
	TR17	TR18	TR19	TR1	TR3	TR2	TR1	TR8	
+6 BDCB6	TR LB								56326
	TR17	TR18	TR19	TR1	TR3	TR2	TR1	TR8	
+7 BDCB7	TR HI								56327
	TR17	TR18	TR19	TR1	TR3	TR2	TR1	TR8	
+8 BDCB8	TDB 1STMS								56328
	0	0	0	0	0	0	0	0	
+9 BDCB9	TDB SEC								56329
	0	0	0	0	0	0	0	0	
+10 BDCBA	TDB HIP								56330
	0	0	0	0	0	0	0	0	
+11 BDCBB	TDB MR								56331
	PR	0	0	PR	PR	PR	PR	PR	
+12 BDCBC	BDR								56332
	B7	B8	B9	B1	B3	B2	B1	B8	
+13 BDCBD	ICR								56333
	ICR	0	0	ICR	ICR	ICR	ICR	ICR	
+14 BDCBE	CRA								56334
	SR HI	SR ADDR	SR ADDR	SR ADDR	SR ADDR	SR ADDR	SR ADDR	SR ADDR	
+15 BDCBF	CRA								56335
	ALPHA	SR ADDR	SR ADDR	SR ADDR	SR ADDR	SR ADDR	SR ADDR	SR ADDR	

# SPRITE-ENTWURFSBLATT





# DATA BECKER BÜCHER



Die überarbeitete und erweiterte 2. Auflage von **VC-20 INTERN** beschäftigt sich detailliert mit Technik und Betriebssystem des VC-20 und enthält ein ausführlich dokumentiertes ROM-Listing, die Belegung der ZEROPAGE und anderer wichtiger Bereiche, übersichtliche Zusammenfassungen der Routinen des BASIC-Interpreters und des VC-20 Betriebssystems, eine Einführung in die Programmierung in Maschinensprache, eine detaillierte Beschreibung der Technik des VC-20 und als Clou einen Original COMMODORE Schaltplan zum Ausklappen! **Damit ist VC-20 INTERN für jeden interessant, der sich näher mit Technik und Maschinenprogrammierung des VC-20 auseinandersetzen möchte.**

VC-20 INTERN, 2. Auflage 1983, ca. 170 Seiten, DM 49,-



Die überarbeitete und erweiterte 2. Auflage von **VC-20 TIPS & TRICKS** enthält eine detaillierte Beschreibung der Programmierung von Sound und Graphik des VC-20, mehr über Speicherbelegung, Speichererweiterung und die optimale Nutzung der einzelnen Speichermodule, BASIC-Erweiterungen zum Eintippen, umfangreiche Sammlung von Poke's und anderen nützlichen Routinen, zahlreiche interessante Beispiel- und Anwendungsprogramme, komplett dokumentiert und fertig zum Eintippen (z.B. Spiele, Funktionsplotter, Graphik Editor, Sound Editor) und vieles andere mehr. **VC-20 TIPS & TRICKS ist eine echte Fundgrube für jeden VC-20 Anwender.**

VC-20 TIPS & TRICKS, 2. Auflage 1983, ca. 230 Seiten, DM 49,-

# DATA BECKER BÜCHER



Jetzt in überarbeiteter und erweiterter 3. Auflage: **64 INTERN** erklärt detailliert Architektur und technische Möglichkeiten des C-64, zerlegt mit einem ausführlich dokumentierten ROM-Listing Betriebssystem und BASIC-Interpreter, bringt mehr über Funktion und Programmierung des neuen Synthesizer Sound Chip und der hochauflösenden Graphik, zeigt die Unterschiede zwischen VC-20, C-64 und CBM 8000 und gibt Hinweise zur Umsetzung von Programmen. Zahlreiche lauffertige Beispielprogramme, Schaltbilder und als Clou: zwei ausführlich dokumentierte Original COMMODORE DIN A3 Schaltpläne zum Ausklappen. **Dieses Buch sollte jeder 64-Anwender und Interessent haben.**

64 INTERN, 3. Auflage 1983, ca. 320 Seiten, DM 69,-



Die überarbeitete und erweiterte 2. Auflage von **64 TIPS & TRICKS** enthält eine umfangreiche Sammlung von POKE's und anderen nützlichen Routinen, Multitasking mit dem C-64, hochauflösende Graphik und Farbe für Fortgeschrittene, mehr über CP/M auf dem C-64, mehr über Anschluß- und Erweiterungsmöglichkeiten durch USER PORT und EXPANSION PORT, sowie zahlreiche ausführlich dokumentierte Programme von der SORT-Routine über zahlreiche BASIC-Erweiterungen bis hin zur 3D-Graphik (alle Maschinenprogramme jetzt mit BASIC-Ladeprogramm!). **64 TIPS UND TRICKS ist eine echte Fundgrube für jeden COMMODORE 64 Anwender.**

64 TIPS & TRICKS, 2. Auflage 1983, ca. 280 Seiten, DM 49,-

# DATA BECKER BÜCHER



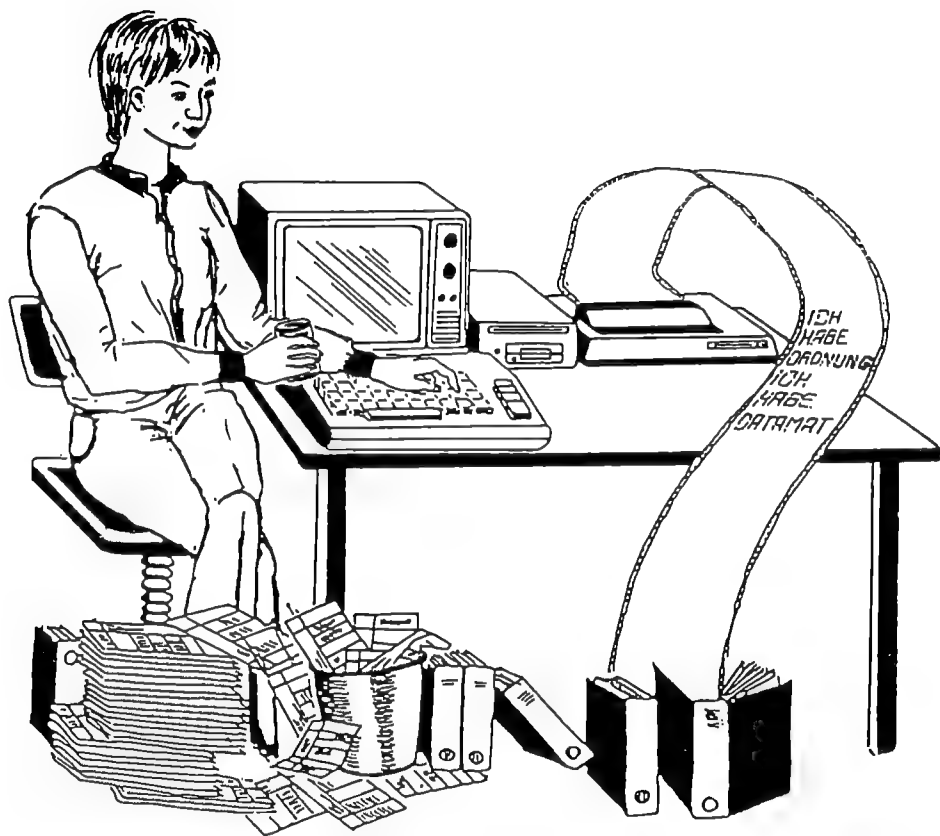
Darauf haben Sie gewartet: Endlich ein Buch, das Ihnen ausführlich und verständlich die Arbeit mit der Floppy VC-1541 erklärt. **DAS GROSSE FLOPPY BUCH** ist für Anfänger, Fortgeschrittene und Profis gleichermaßen interessant. Sein Inhalt reicht von der Programmspeicherung bis zum DOS-Zugriff, von der sequentiellen Datenspeicherung bis zum Direktzugriff, von der technischen Beschreibung bis zum ausführlich dokumentierten DOS Listing, von den Systembefehlen bis zur detaillierten Beschreibung der Programme der Test/Demodiskette. Exakt beschriebene Beispiel- und Hilfsprogramme ergänzen dieses neue Superbuch. **Mit dem GROSSEN FLOPPY-BUCH meistern Sie auch Ihre Floppy.** **DAS GROSSE FLOPPY BUCH**, 1983, ca. 250 Seiten, DM 49,-



Wer besser und leichter in BASIC programmieren möchte, der braucht dieses neue Buch. **64 FÜR PROFIS** zeigt, wie man erfolgreich Anwendungsprobleme in BASIC löst und verrät Erfolgsgelheimnisse der Programmierprofis. Vom Programmentwurf über Menüsteuerung, Maskenaufbau, Parameterisierung, Datenzugriff und Druckausgabe bis hin zur Dokumentation wird anschaulich mit Beispielen dargelegt, wie gute BASIC-Programmierung vor sich geht. Fünf komplett beschriebene, lauffertige Anwendungsprogramme für den C-64 illustrieren den Inhalt der einzelnen Kapitel beispielhaft. **Mit 64 FÜR PROFIS lernen Sie gute und erfolgreiche BASIC-Programmierung.** **64 FÜR PROFIS**, 1983, 220 Seiten, DM 49,-

# DATAMAT

Der Datenautomat für den Commodore 64



DATAMAT – Das ist Software MADE IN GERMANY

ausgereift, leistungsstark und preiswert

Ein DATA BECKER Produkt

geschrieben von Wolfgang Schellenberger

\*\*\*\*\*

## DATAMAT - Systembeschreibung

\*\*\*\*\*

### Warum brauchen Sie DATAMAT ?

Sicher haben auch Sie eine Fülle von Unterlagen, die Sie verwalten müssen. Dazu haben Sie bis jetzt immer Karteikarten, Aktenordner und Registraturen verwendet. Oder Sie haben eine umfangreiche Schallplatten, Bücher oder Videokassettenammlung. Oder Sie haben ein großes Notizbuch mit den Adressen aller Ihrer Bekannten.

Und nun suchen Sie etwas. Einen bestimmten Titel auf einer Schallplatte, eine bestimmte Adresse.

Und genau jetzt setzt DATAMAT ein. Mit DATAMAT können Sie :

- feststellen, auf welcher Schallplatte sich ein bestimmter Titel befindet
- feststellen, wer nächste Woche Geburtstag hat.
- eine sortierte Liste aller Mitglieder Ihres Vereins erstellen
- herausfinden, ob Sie einen bestimmten Artikel noch am Lager haben
- feststellen, wer diesen Artikel liefert

Die Einsatzmöglichkeiten von DATAMAT sind praktisch unbegrenzt. Beenden Sie die lange Sucherei nach irgendwelchen Informationen. Mit DATAMAT hat die Wühlerei ein Ende.

### Natürlich auch für Anfänger !

Um mit DATAMAT zu arbeiten, benötigen Sie keinerlei Vorkenntnisse. Modernste Menüsteuerung macht es möglich, daß jeder mit diesem Programmpaket umgehen kann. In allen Teilen des Programms werden Sie von leicht verständlichen, deutschen Fragen und Kommandos geführt. Das umfangreiche deutsche Handbuch, in dem Sie schrittweise anhand einer Adressverwaltung in die Arbeit mit DATAMAT eingeführt werden, erklärt alle eventuell noch anstehenden Fragen.

# \*\*\*\*\* DATAMAT - Systembeschreibung \*\*\*\*\*

## Was ist nun eine "Dateiverwaltung" ?

Der Begriff "Datei" ist leicht erklärbar. Stellen Sie sich vor, Sie haben einen Karteikasten mit vielen gleich aufgebauten Karteikarten, auf denen z.B. Ihre Kunden oder Bekannten eingetragen sind. Genauso werden Ihre Adressen nun auf Diskette abgespeichert. Ihre Karteikarte heißt nun "Datensatz" . Jede Adresse ist ein Datensatz und besteht aus "Datenfeldern". Das sind die Einträge auf der Karteikarte, also z.B. Name, Vorname, Straße, Geburtsdatum, Hobbies u.s.w. Alle Datensätze (Karteikarten) zusammen bilden die "Datei". In dieser Datei können Sie nun einzelne Datensätze löschen, ändern, neue hinzufügen, nach bestimmten Merkmalen suchen, sortieren oder Listen ausdrucken.

## Sie bestimmen den Aufbau Ihrer Datei !

Im Gegensatz zu Individuallösungen können Sie mit DATAMAT Ihre Datei völlig frei konfigurieren. Das bedeutet, daß Sie die unterschiedlichsten Arten von Daten erfassen können. Sie bestimmen den Aufbau der Eingabemaske völlig frei und können so von Adressen über Videokassetten bis Küchenrezepten alles verwalten, was Ihnen Spaß macht.

Und so könnten z.B. Eingabemasken aussehen :

<div style="border: 1px solid black; padding: 5px;"> <p>Vorname : <u>                  ↑                  ↓                  </u></p> <p>Name : <u>  ↑  </u></p> <p>Straße : <u>  ↑  </u></p> <p>PLZ : <u>          ↑  </u></p> <p>Wohnort : <u>  ↑  </u></p> <p>Telefon : <u>  ↑  </u></p> <p>Hobbies : <u>  ↑  </u></p> </div>	<p style="text-align: right;">L A G E R    Schmitz KG</p> <p>Artikelnummer : <u>          ↑                  ↓                  </u></p> <p>Bezeichnung : <u>  ↑  </u></p> <p>Bestand : <u>                  ↑                  ↓                  </u></p> <p>geliefert am : <u>          ↑                  ↓                  </u></p> <p>EK pro Stueck : <u>          ↑                  ↓                  </u> DM</p> <p>VK pro Stueck : <u>          ↑                  ↓                  </u> DM</p> <hr/> <p>Lieferant : <u>  ↑  </u></p> <p>Straße : <u>  ↑  </u></p> <p>PLZ : <u>          ↑                  ↓                  </u> Ort : <u>  ↑  </u></p> <p>Tel. : <u>  ↑  </u></p>
--	---

Die Pfeile markieren den Anfang und das Ende der Eingabefelder. Die Länge der Datensätze ergibt sich aus der Summe der Länge aller Eingabefelder. In unseren Beispielen beträgt diese Summe 128. Addieren Sie nun 1, und Sie erhalten die Länge der Datensätze, wie sie auf der Diskette abgespeichert werden. Wieviele Datensätze nun auf Ihrer Diskette Platz haben, hängt von der Länge der Datensätze ab.

\*\*\*\*\*

## DATAMAT - Systembeschreibung

\*\*\*\*\*

Zum Anlegen der Bildschirmmaske steht Ihnen der volle Bildschirmeditor des Commodore 64 zur Verfügung. Weiter haben Sie Funktionen wie Zeilen einfügen, löschen, Farbe ändern u.s.w. zur Verfügung. Diese Maske können Sie natürlich jederzeit ändern.

### Eingeben, Suchen, Ändern, Löschen von Daten

In diesem Programmteil erfassen Sie Ihre Daten mit Hilfe der Bildschirmmaske, die Sie vorher erstellt haben. Sie können jederzeit vorhandene Datensätze natürlich ändern oder wieder löschen. Ebenfalls können Sie nach bestimmten Informationen suchen, z.B. alle Leute mit dem Nachnamen Schmidt, die in Düsseldorf wohnen. Die gefundenen Datensätze können Sie dann verändern. Die mittlere Zugriffszeit liegt hier bei ca. 4,5 Sekunden, egal auf welchen Datensatz sie zugreifen und wieviele Datensätze Sie bereits eingegeben haben.

### Sortieren, Drucken der Daten

Dieser Programmteil zeigt die wahren Stärken des DATAMAT. Wieder mit Hilfe der Eingabemaske geben Sie die Selektionskriterien und Sortierstufen ein. Zuerst selektieren Sie, d.h. Sie bestimmen z. B., daß Sie alle Leute auswählen wollen, die Müller heißen, im Postleitzahlgebiet 4500 bis 5678 wohnen, kein Auto besitzen und größer als 1,70 Meter sind. Dann sortieren Sie, so kann Ihre Liste z.B. zuerst nach den Postleitzahlen, bei gleicher Postleitzahl nach den Straßen und bei gleicher Straße nach der Hausnummer sortiert werden. Sie können festlegen, ob aufsteigend oder fallend sortiert werden soll. Mit diesen Kriterien können Sie Listen oder Etiketten drucken, deren Aussehen Sie völlig frei bestimmen können. Bei Listen können Sie Überschriften festlegen und Spalten summieren.

### DATAMAT - die wichtigsten Daten in kurze

- Ihre Karteikarte kann bis zu 50 verschiedene Einträge enthalten
- Bis zu 2000 Karteikarten können Sie verwalten
- Die Zugriffszeit liegt bei ca. 4,5 Sekunden mit einer 1541 - Floppy
- Sie können nach jedem Eintrag suchen
- Sie können nach jedem und beliebig vielen Einträgen selektieren und sortieren in soviel Stufen, wie Sie Eingabefelder haben
- Sie können Etiketten drucken
- Sie können Listen drucken, die Überschriften auf jeder Seite haben und Spalten summieren
- Sie können jede Art von Daten verwalten, von Kunden bis zu Kochrezepten

DATAMAT - EIN DATEIVERWALTUNGSGROGRAMM FÜR DEN COMMODORE 64, DAS ÜBERZEUGT.

Mit DATAMAT haben wir das erste Programm in der neuen Reihe der

### **DATA BECKER PROGRAMME**

vorge stellt. Ziel dieser neuen Reihe ist es, den Anwendern des COMMODORE 64 für wenig Geld professionelle Programme zugänglich zu machen. Nur in einem Punkt haben wir Kompromisse gemacht: beim Preis. Jedes der Programme kostet trotz der außergewöhnlichen Leistungsmerkmale nur

**DM 99,-** (unverbindl. Preisempfehlung incl. 14% MwSt.)

Ab Oktober/November '83 sind auch die folgenden Programme erhältlich:

## **PROFIMAT**

Ein Spitzenpaket für Maschinenspracheprogrammierer. PROFIMAT enthält nicht nur unseren komfortablen Maschinensprache-Monitor PROFI-MON, sondern auch PROFI-ASS, einen sehr leistungsfähigen Assembler für den COMMODORE 64. PROFI-ASS bietet unter anderem formatfreie Eingabe, komplette Assemblerlistings, ladbare Symboltabellen (Labels), verschiedene Möglichkeiten zur Speicherung des erzeugten Maschinencodes, redefinierbare Symbole, eine Reihe von Pseudo-Codes (Assembleranweisungen), bedingte Assemblierung und die Möglichkeit zur Erzeugung von Assemblerschleifen. PROFIMAT kostet komplett nur DM 99,-.

## **BASIC 64**

Dieser neue 1-Pass-BASIC-Compiler macht Ihre Programme bis zu 10mal schneller. Er erzeugt direkten Maschinencode, der beliebig im Speicher platzierbar ist. BASIC 64 unterstützt Fließkommaarithmetik, Stringverwaltung und den gesamten 64er Befehlssatz bis auf FRE, TAB, SPC, ON X GOTO/ GOSUB, mehrdimensionale Felder und Klammerrechnung. Ein Superknüller für nur DM 99,-.

## **PASCAL 64**

Endlich ein PASCAL für den 64er. PASCAL 64 hat einen großen Befehlssatz mit allen wesentlichen Standardbefehlen und enthält auch Dateiverwaltungsbefehle. AOS-Arithmetik real und integer. Kein eigener Editor erforderlich, da im Commodore Editor-Modus eingegeben werden kann. PASCAL 64 ist sehr schnell, da echter Maschinencode erzeugt wird, und kostet komplett mit ausführlichem Handbuch nur DM 99,-.

## **SUPERGRAPHIK 64**

Die neueste Version unserer beliebten SUPERGRAPHIK enthält jetzt über 30(!) Befehle zur Ausnutzung der fantastischen Möglichkeiten, die der 64 mit hochauflösender Graphik und Farbe bietet. Mit SUPERGRAPHIK 64 können Sie Punkte, Linien und Kreise ziehen, SPRITES definieren und manipulieren, Farben setzen, komplette Graphikbildschirme auf Diskette abspeichern bzw. laden und vieles andere mehr. Ergänzt wurde die SUPERGRAPHIK 64 zusätzlich um SUPERSOUND, eine neue Befehlserweiterung zur Nutzung der hervorragenden Soundmöglichkeiten des 64. Mit SUPERGRAPHIK 64 machen Sie mehr aus Ihrem 64er, und das für nur DM 99,-.

## **TEXTOMAT**

Ein außergewöhnliches Textverarbeitungsprogramm. Bis zu 255 Zeichen pro Zeile mit horizontalem Scrolling. Texte bis zu 24000-Zeichen, Textbaustein-Verarbeitung, umfangreiche Formatierungsmöglichkeiten, Schnittstelle zu DATAMAT für Rundschreiben und Serienbriefe und vieles andere mehr. TEXTOMAT ist komplett in Assembler geschrieben und sehr schnell. TEXTOMAT ist natürlich in deutsch, mit deutscher Bedienungsführung und kostet mit ausführlichem Handbuch nur DM 99,-.



## **DATAMAT**

Eine universelle Dateiverwaltung, die Sie von der Adressverwaltung über die Mitgliederverwaltung bis zur Lagerbuchführung auf vielfältigste Weise nutzen können. Die frei gestaltbare Eingabemaske kann bis zu 50 Felder, max. 40 Zeichen pro Feld und max. 253 Zeichen pro Datensatz enthalten. Bis zu 2000 Datensätze pro Diskette sind möglich. Nach allen Feldern kann sortiert und selektiert werden, sogar nach mehreren gleichzeitig. Auswertungen können als Listen und als Etiketten gedruckt werden. Ein Superprogramm, das zu jedem 64er gehören sollte. Komplett mit ausführlichem Handbuch nur DM 99,-.

## **KONTOMAT**

Ein Einnahme-Überschußprogramm nach § 4 (3) EStG mit Kassenbuch, Bankkontenüberwachung, automatischer Steuerbuchung (Brutto u. Netto), AfA Tabellenerstellung, Kontenblättern & Journal, Ermittlung der USt.-Voranmeldungswerte und Monats- und Jahresrechnung. KONTOMAT ist voll parameterisiert (Firmendaten, Steuersätze, Konten, Buchungstexte) und läßt sich damit an Ihre Bedürfnisse anpassen. KONTOMAT ist geeignet für alle Selbständigen und Gewerbetreibenden, die nicht laut HGB zur Buchführung verpflichtet sind. Komplett mit ausführlichem Handbuch nur 99,-.

## **FAKTUMAT**

Eine Sofortfakturierung mit integrierter Lagerbuchführung. Die Kunden- und Artikelstammdatei ist voll pflegbar. Steuersätze, Maßeinheiten und Firmendaten sind individuell anpaßbar. Schneller Diskettenzugriff auf Kunden- und Artikeldaten. Schnittstelle zur Textverarbeitung. Komplett mit ausführlichem Handbuch nur DM 99,-.

## **SYNTHIMAT**

Mit diesem Superprogramm verwandeln Sie Ihren 64er in einen professionellen, polyphonen, dreistimmigen Synthesizer, mit dem Sie über die Tastatur ganze Akkorde spielen können. Zu den unglaublich vielen Möglichkeiten dieses Programms gehört auch die „Bandaufnahme-/Wiedergabe“ direkt auf bzw. von Diskette. Verwandeln Sie Ihren 64er für wenig Geld in eine Super-Musikmaschine mit SYNTHIMAT. Komplett mit ausführlichem Handbuch nur DM 99,-.

DATA BECKER PROGRAMME erhalten Sie dort, wo Sie auch DATA BECKER BÜCHER bekommen:

- im COMMODORE-Fachhandel
- in großen Kauf- und Warenhäusern
- in Fachbuchhandlungen

oder direkt von DATA BECKER. Vertrieb in der Schweiz über THALI AG und in Österreich über Fachbuchcenter ERB.

Nie wieder „zu Fuß“ programmieren müssen!

**NEU:** Jetzt auch Commodore

# MASTER

**Das professionelle Programm-Entwicklungssystem für  
CBM 8032 und 8096, Commodore 64, 600 und 700**

„Ich habe doch einen leistungsfähigen Computer mit einem komfortablen BASIC Interpreter. Wofür brauche ich dann überhaupt noch ein Programm-Entwicklungssystem?“ - So werden Sie zurecht fragen. Ihr Commodore Computer bietet heute für erstaunlich wenig Geld in der Form eines Tischcomputers Leistungen, wie sie noch vor Jahren nur mit Großrechnern möglich waren. Leider aber enthalten Betriebssystem und BASIC Ihres Commodore nicht die Elemente, die man braucht, um auf einfache Weise Programme zu erstellen, die anwenderfreundlich, schnell, sicher, leicht änderbar und aufwärtskompatibel zu größeren Computern sind. Dies gilt praktisch für fast alle Computer, vom Micro bis zum Großrechner. Besonders im Großrechnerbereich hat man sich deshalb als Hilfsmittel Software-Werkzeuge geschaffen, die eine einfache Erstellung anspruchsvoller Programme ermöglichen. MASTER ist ein solches Werkzeug, das sich in der Leistung an den Vorbildern aus dem Großrechnerbereich orientiert, aber im Preis Ihrem Commodore Computer angeglichen ist.

Im Großrechnerbereich ist eine Programmierung ohne entsprechende Hilfsmittel praktisch undenkbar. Jetzt können auch Commodore-Programmierer auf derartigen Komfort bei der Programmierung zurückgreifen. MASTER ist nicht „schon wieder eine BASIC-Erweiterung“, sondern ein ausgereiftes geschlossenes Konzept zur komfortablen Programmerstellung. MASTER macht aus dem COMMODORE BASIC eine leistungsfähige Sprache und enthält alle dafür wesentlichen Elemente:

- **BILDSCHIRM-VERWALTUNG**  
zur raschen Erstellung komfortabler Bildschirmmasken
- **ISAM-DATEIVERWALTUNG**  
für schnellen Datenzugriff und effiziente Dateiverwaltung
- **DRUCK-GENERATOR**  
zum einfachen Erstellen und Ausdrucken beliebiger Ausgabemasken
- **PROGRAMMSCHUTZ**  
durch NOLIST-Modus und individuelle Schlüssel (auf Wunsch)
- **MEHRFACHGENAUE ARITHMETIK**  
Rechnen mit 22 Stellen Genauigkeit
- **BASIC-ERWEITERUNGEN**  
Toolkit-Funktionen, nützliche Zusatzbefehle und in der 64er Version das komplette BASIC 4.0

Selbstverständlich ist MASTER ausgereift und ausführlich in Deutsch dokumentiert. MASTER kostet kaum mehr als ein Programmiertag und ist etliche Mannmonate wert. Über 5000 Programmierer nutzen bereits die Vorteile von MASTER. Machen Sie mit!

MASTER ist ein Produkt der französischen Firma Micro Application und wird in Deutschland von DATA BECKER über den Commodore-Fachhandel vertrieben.

**VC-20 · COMMODORE 64 · EXECUTIVE**

**DA STEHT ALLES DRIN!**

# **VC-INFO**

**3/83 ist da!**

Der neue, 80(!)seitige Katalog rund um den VC-20, COMMODORE 64 und den neuen COMMODORE EXECUTIVE, mit den neuesten Software-Hits aus aller Welt, interessantem Zubehör, vielseitigen Peripheriegeräten, neuen Superbüchern, Programmiertips & Tricks und der großen Übersichtstabelle »Was läuft womit«. Das VC-INFO 3/83 erhalten Sie gegen DM 3,- in Briefmarken.

***IHR GROSSER PARTNER  
FÜR KLEINE COMPUTER***

## ***DATA BECKER***

Merowingerstraße 30 · 4000 Düsseldorf 1  
im Hause AUTO BECKER · Telefon 0211/310010

**VC-20 · COMMODORE 64 · EXECUTIVE**

**VC-20 · COMMODORE 64 · EXECUTIVE**

**EXECUTIVE · COMMODORE 64 · VC-20**

